QUT
Library

GYAN: A Methodology for Rule Extraction from Artificial Neural Networks

By

Richi Nayak
B.Eng. (G.E.C. Bilaspur, India), M.Eng. (University of Roorkee, India)

A Dissertation Submitted in Partial Fulfillment of the
Requirement for the Degree of

Doctorate of Philosophy

Gardens Point Stack
A20487940B
GYAN : a methodology
for rule extraction
from artificial neural
networks

Machine Learning Research Centre
School of Computing Science
Faculty of Information Technology
Brisbane, Queensland, Australia
1999

QUEENSLAND UNIVERSITY OF TECHNOLOGY

DOCTOR OF PHILOSOPHY THESIS EXAMINATION

CANDIDATE NAME	***Richi Nayak***
CENTRE/RESEARCH CONCENTRATION	***Machine Learning Research Centre***
PRINCIPAL SUPERVISOR	***Professor Joachim Diederich***
ASSOCIATE SUPERVISOR(S)	***Dr Frederic Maire***
THESIS TITLE	***Gyan: A Methodology for Rule Extraction from Artificial Neural Networks***

Under the requirements of PhD regulation 9.2, the above candidate was examined orally by the Faculty. The members of the panel set up for this examination recommend that the thesis be accepted by the University and forwarded to the appointed Committee for examination.

Name: PROF J DIEDERICH ..... Signature QUT Verified Signature
Panel Chairperson *(Principal Supervisor)*

Name: Dr Chris Ho-Stuart ..... Signature QUT Verified Signature
Panel Member

Name: Dr Alan B. Tickle ..... Signature QUT Verified Signature
Panel Member

Name: ..... Signature QUT Verified Signature
Panel Member

Under the requirements of PhD regulation 9.15, it is hereby certified that the thesis of the above-named candidate has been examined. I recommend on behalf of the Thesis Examination Committee that the thesis be accepted in fulfillment of the conditions for the award of the degree of Doctor of Philosophy.

Name Dr Alan B Tickle Signature... QUT Verified Signature Date 26 June 2000
Chair of Examiners (External Thesis Examination Committee)

Key words

artificial neural networks, rule extraction, first order logic, data mining, knowledge acquisition, machine learning, inductive learning

Abstract

Artificial neural network (ANN) learning methods provide a robust and non-linear approach to approximating the target function for many classification, regression and clustering problems. ANNs have demonstrated good predictive performance in a wide variety of practical problems. However, there are strong arguments as to why ANNs are not sufficient for the general representation of knowledge. The arguments are the poor comprehensibility of the learned ANN, and the inability to represent explanation structures.

The overall objective of this thesis is to address these issues by: (1) explanation of the decision process in ANNs in the form of symbolic rules (predicate rules with variables); and (2) provision of explanatory capability by mapping the general conceptual knowledge that is learned by the neural networks into a knowledge base to be used in a rule-based reasoning system.

A multi-stage methodology GYAN is developed and evaluated for the task of extracting knowledge from the trained ANNs. The extracted knowledge is represented in the form of restricted first-order logic rules, and subsequently allows user interaction by interfacing with a knowledge based reasoner. The performance of GYAN is demonstrated using a number of real world and artificial data sets. The empirical results demonstrate that: (1) an equivalent symbolic interpretation is derived describing the overall behavior of the ANN with high accuracy and fidelity, and (2) a concise explanation is given (in terms of rules, facts and predicates activated in a reasoning episode) as to why a particular instance is being classified into a certain category.

Contents

List of Figures

List of Tables

Glossary of terms and abbreviations

AI artificial intelligence.

ANN artificial neural network.

BT (BpTower) a feedforward cascade type network that employs *gradient descent* for learning a single layer network.

C4.5 a decision tree learning algorithm.

CC cascade correlation network.

CEBPN (constrained error back propagation network) a special class of feedforward networks that utilizes *local functions* in the construction of hidden layers.

CNF (conjunctive normal form) an expression can be represented as a conjunction of disjunctions, such as $disnjunct_1 \wedge disjunct_2 .. \wedge disjunct_n$. Each individual disjunction can be of arbitrary length, such as $a_1 \vee a_2 .. \vee a_k$.

CSN connectionist semantic network.

class'n (classification) mismatch a measure of rule accuracy, given as the ratio of the incorrectly classified instances to the total number of instances.

comprehensibility a measure of symbolic interpretation of a generated rule set, given as the number of predicate rules, the number of entities, and the number of conjunctive expressions, etc.

DNF (disjunctive normal form) an expression can be represented as a disjunction of conjunctions, such as $conjunct_1 \vee conjunct_2.. \vee conjunct_n$. Each individual conjunction can be of arbitrary length, such as $a_1 \wedge a_2.. \wedge a_k$.

discretisation a process that quantizes the numeric data $[min_i, \ldots, max_i]$ into a number of intervals, and maps each interval to a discrete value/symbol.

Foil a first order inductive inductive learning algorithm.

fidelity a measure of the agreement between the ANN and the extracted rule set.

functional dependencies If a variable can be presented with another variable to represent a function or a rule, the variables are called dependent. For example, if a person is of a N nationality then the person speaks a L language, can be easily represented by $langauge(P, L) \Leftarrow nationality(P, X)$.

Gyan a multi-stage methodology to generate predicate rules from neural networks, developed in this thesis.

inferencing a process by which an inference engine reaches to a conclusion.

LAP a decompositional rule extraction technique that derives symbolic rules from analyzing (search and test) weight parameters in the feedforward networks.

lgg (least general generalization) if a sentence C θ-subsumes a sentence D, such that $C\theta \subseteq D$, C is the least general generalization of D under θ-subsumption. (see also θ-subsumption)

ML machine learning.

Network architecture the number of input (I), hidden (H) and output (O) nodes in ANNs.

PBT pruned BpTower networks.

PCC pruned cascade correlation networks.

predicate (pred') rules see restricted first order rules.

RMSE root mean square error.

RULEX a decompositional rule extraction technique that directly interprets each *local hidden unit (weights)* in CEBPN into symbolic rules.

RuleVI a pedagogical rule extraction technique that generates a rule set by repeatedly changing antecedents of the training instances, querying the trained ANN, and examining the network's response.

reasoning see inferencing.

restricted first-order rules first-order rules (*i.e.* rules with variables and n-ary predicates) with the following limitations: (1) any variables occurring in antecedents of a rule must occur in the consequent of the rule and (2) recursively defined predicates and infinite terms are not allowed.

SHRUTI a connectionist knowledge based reasoning system to efficiently enable dynamic inferencing based on the synchronous firing of constituent nodes in distinct phases.

sparse-coded representation each value of the discrete (categorical) attribute with n possible values is represented by an n-bit binary string, with only one bit carrying a value of one corresponding to the attribute's value.

θ-subsumption the subsumption method to eliminate all sentences that are subsumed by a least general sentence in the extracted rule set.

weight decay a regularisation term included in the error function to be minimized during ANN training.

Statement of original authorship

The work contained in this thesis has not been previously submitted for a degree or diploma at any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: QUT Verified Signature

Date 10/07/2000

Acknowledgments

I will take this opportunity to express my gratitude to Prof. Joe Diederich, my thesis supervisor, for his guidance, help and encouragement during the course of this work. It has been a privilege and a pleasure to work with him. He has provided the research environment that makes this thesis possible.

I would like to thank Assoc. Prof. C. Szyperski to helping me in the early stages of my research. I would also like to thank Assoc. Prof. G. Mohay and the administrative and technical staff at the Faculty of Information Technology, QUT for their generous support and assistance throughout these years. Moreover, the financial support provided by the overseas postgraduate research scholarship (for the initial first year) and by the Machine Learning Research Centre (MLRC) (throughout this study) is greatly appreciated.

I am indebted to Russell Cable for his efforts and patience in proof-reading my thesis. Further thanks are due to many members at the MLRC for their stimulating discussions and friendly working environment, especially Ross Hayward, Hussein Abbass, Alan Tickle, Chris Ho-Stuart, Frederic Maire, Robert Andrews, Stephan Chalup, M. T. Wong, Michael Towsey, Justin Lee, and Will Bligh. I would especially like to thank Ross Hayward, Linus Dillon and Robert Andrews for providing me with the computer programs.

I am grateful to my mother Mrs. Pushpa Panchaity and my parents-in-law Prof. G. C. Nayak and Mrs Indra Nayak for their love and constant support, without them this work would not have been possible. Last and least, I am

grateful to my husband Anurag for his enduring support, love, patience and advice that truly made this research work to be realized.

This thesis is dedicated to my late father Dr. U. K. Panchaity, who always encouraged me to pursue a career in research and teaching.

Chapter 1

Introduction

Since their revival in the early 1980s, artificial neural networks (ANNs) have been extremely valuable for learning from examples and making predictions for unseen examples. ANNs have been successfully applied to a wide range of pattern recognition and function approximation problems [Murray, 1992; Simpson, 1996; Browne, 1997]. However, there are strong arguments as to why ANNs are not sufficient for the general representation of knowledge (the desired function).

One of the arguments is the inability to explain the decision process, not readily yielding coherent explanations as to why certain data is classified into a particular category. ANNs do not have explicit, declarative knowledge structures that allow the representation of explanation structures such as reasoning paths, explanation of expectation failure etc [Diederich, 1992]. The trained ANN is considered very much a *'black box'* solution to the underlying problem, as its structure and reasoning is relatively inaccessible to higher level reasoning. This limits the range of possible explanation methods significantly [Diederich, 1992]. Therefore ANNs must overcome their limitations as representational systems to gain an even wider degree of user acceptance and to enhance their overall utility as learning and generalization tools.

1.1 Rule extraction from ANNs

Symbolic machine learning systems have an explicit, declarative representation of knowledge about a problem domain that enables a user explanation about *how and why* a conclusion has been reached. A user friendly system such as this is widely required in practical applications. The successful application of ANN methods and systems in fields such as commerce, engineering, medicine and science, offers a clear testament to the capability of the ANN paradigm [Andrews *et al.*, 1995]. However, ANNs require an explanation mechanism similar to those in symbolic machine learning systems. This will enable users to understand its decision process and hence gain more widespread user acceptance.

The problem of understanding why a trained ANN makes a decision has a long history in the field of connectionist modeling. Knowledge in an ANN is distributed across the network and embedded in the form of numerical weight parameters. One promising approach to this problem is to map the weighted parameters into a symbolic description. The symbolic interpretation of knowledge embedded in the trained ANN is an approach to overcome the *'black box'* problem. This type of reformulation known as *rule extraction* can explain the network behavior and facilitate transfer of learning as well. ANN techniques identify the dependencies existing in data and predict future behavior, and rule extraction techniques provide the means to comprehend the decision process of ANNs.

The need for understanding an ANN's functional behavior in critical domains has led to a number of approaches that try to explain the ANN's decision process with various types of symbolic rule formalism. The rules that are extracted from ANNs after training are usually in propositional form. However, there are limitations inherent in propositional logic. For example, a rule set consisting of purely local rules that attempt to explain a problem that is non-local in nature are not sufficient. The (local or propositional) rule set extracted from the network may be accurate in terms of data points

classified and overly specific, however the actual problem description will not be revealed such as X = Y where X and Y are two attributes in the data set. There must be an approach of mapping such a set of propositional rules to a rule set expressed in first, or higher order logic where the non-localness is made explicit. The mapping of propositional rules to generalized rules (with variables and n-ary predicates) provides a language suitable for describing the knowledge embedded in ANNs [Nayak and Diederich, 1998; Visser *et al.*, 1998]. The predicate rules allow learning of general rules as well as learning of internal relationships among variables.

1.2 Rule insertion

Connectionist semantic networks (CSNs) are able to encode a wide variety of data such as rules with variables, stored facts, and type-hierarchy [Shastri, 1988]. The emphasis of CSNs is on knowledge representation, reasoning strategies and the ability to explain through a large number of highly interconnected but relatively simple processing elements [Gűsgen and Hőlldobler, 1992]. CSNs are suited for user explanation because of their explicit structure, such as the inheritance hierarchy [Diederich, 1992]. In most of the CSNs, for instance in the SHRUTI connectionist semantic network [Shastri and Ajjanagadde, 1993], the rules and facts (comprising the knowledge base) are hardwired, and there are no mechanisms provided to adapt the knowledge base. Provision of a learning component for such artificial reasoning systems is essential [Dillon, 1997].

The limitation of *representational power and declarative ability* in ANNs can be handled by inserting the attained symbolic form of rules from the trained network (and processed into (a subset of) first order logic) into connectionist semantic networks like SHRUTI. Since the two approaches address different levels of knowledge representation and problem solving, combining their strength in an integrated environment is justified. The symbolic rules processed from a trained ANN can be fed into a rule-based connectionist rea-

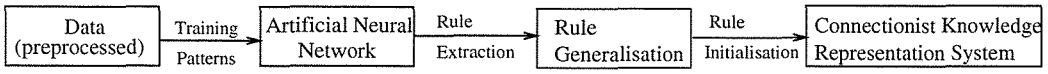


Figure 1.1: The Gyan methodology

soning system. This system then performs a similar task to the ANN but with an explanatory capability. The explanation is gained through operational predicates, rules, and facts activated in a reasoning episode. As Diederich [1992] argues that explanation is not just a simple replay of activation patterns but a partial or full reproduction of inference states which makes the structural relations explicit that have been used during inference. Additionally, in case of partially specified queries posed during a reasoning episode, the unknown values (or variables) are supplemented by the reasoner according to the stored knowledge. This also alleviates to some extent the problem of limited or incomplete data.

1.3 Scope of the thesis

The thesis is driven by two underlying goals: (1) explanation of the decision process in ANNs in the form of symbolic rules (predicate rules with variables); and (2) provision of explanatory capability by mapping the general conceptual knowledge that is learned by ANNs into a knowledge base to be used in a rule-based reasoning system.

A domain-independent multi-stage methodology GYAN[1] (Figure 1.1) is developed and evaluated for the task of representing knowledge from the trained ANNs in the form of restricted first-order rules[2]. The objective is to apply GYAN to a combination of pattern recognition and decision making problems.

[1] GYAN is a Sanskrit word which refers to the knowledge or awareness gained by examples/experience.

[2] Recursively defined predicates and infinite terms are not allowed in this knowledge representation. There are also restrictions on the appearance of variables in rules to satisfy the constraints imposed by the SHRUTI connectionist reasoning system. This type of rules are also called 'predicate rules with variables'.

The GYAN methodology derives an equivalent symbolic interpretation describing the overall behavior of the ANN, and also gives a concise explanation (in terms of activated rules, facts and predicates in a reasoning episode) as to why a particular instance is being classified into a certain category. GYAN can be successfully applied to *knowledge discovery and data mining tasks* and to *automate knowledge acquisition tasks.*

1.4 Thesis organization

A brief overview of this thesis is as follows:

Chapter 2 provides background material for the generation of symbolic rules from ANNs. This chapter provides a link between the methodology presented in the thesis and work in the field of inductive learning. Previous researchers have done much work in the area of learning a set of rules from a set of training examples. This chapter describes three types of techniques for empirical learning tasks: rule extraction from ANNs; symbolic propositional inductive learning (learning decision trees and logical concept definitions from examples); and the first-order inductive learning techniques. This chapter also contains an extensive survey of techniques (published to date) for rule-extraction from ANNs. The background material on rule extraction from ANNs is included to enable a better understanding of the contributions of the methodology presented in subsequent chapters.

Chapter 3 introduces the basic ideas in the GYAN methodology, *i.e.* the integration of neural learning and rule-processing. This chapter presents a framework for the integration and explains the whole process with a simple example. This chapter also briefly reviews a few (representative) connectionist semantic networks.

Chapter 4 presents the GYAN methodology that generates the restricted first-order logic rules from the trained ANN, and subsequently allows

user interaction by interfacing with a knowledge based reasoner. The GYAN methodology is the main contribution of this thesis. It consists of many stages and includes many components. The framework has several strong points. The first one is the ability to map the trained ANN into a set of predicate rules. The predicate rules can then be processed by a connectionist reasoning system to provide an explanation of the decision process in the trained ANN. The important aspect is the mapping of extracted propositional rules into the predicate rules with variables. This mapping is done by a procedure containing a set of post-processing and generalization rules and acts as an interface between the trained ANN and any inference engine. Another important point is the identification of important variables for the rule-extraction process by pruning irrelevant nodes and links from the trained ANN. The distinguishing feature of the pruning algorithm is the identification of relevant connections from the input nodes to the hidden/output nodes based on the magnitude of their weights, and maintaining the predictive accuracy of the ANN after pruning. Another strong point is the applicability of predicate rule-extraction to a broad class of ANN architectures.

Chapter 5 lays the foundation for the experimental analysis of the GYAN methodology. This chapter first deals with data representation steps to prepare data according to relevant classifiers in GYAN. This chapter briefly describes all the application domains that are used to evaluate GYAN. Chapter 5 also contains the evaluation criteria to present the outcome of this methodology.

Chapter 6 presents a series of experiments that empirically evaluate the GYAN methodology in the context of classification learning tasks. The purpose of the experiments is to evaluate the effectiveness of GYAN along the dimensions of explanatory power, accuracy, fidelity and comprehensibility. For each of the data sets, the results obtained using GYAN compare favorably with those reported by other authors, and those obtained with other algorithms (FOIL and C4.5). The successful applica-

QUT
Library

tion and competitive results obtained by GYAN for various problem domains demonstrate its effectiveness in real-life problems (such as 'Queensland Rail' and 'Remote Sensing'), in fairly large size problems (in terms of number of attributes such as 'Breast Cancer', 'Moral Reasoning', 'Voting' and 'Mushroom'), and in continuous-valued problem domains (such as 'Cleveland heart disease'). Importantly, the agreement in the results obtained by different components at any stage of GYAN demonstrates the efficiency of the framework.

Chapter 7 concludes the thesis with contributions, limitations of the thesis, and suggestions for further work.

Appendix A describes some of the predicate rules generated by GYAN for all the problem domains discussed in chapter 5.

Chapter 2

Background and related work

The main focus of this chapter is to provide background material for the remainder of the thesis. The chapter starts with the description of inductive learning and summarizes the generalization operators to derive general rules from the given examples. The description of these operators is essential as they are used further in the thesis to compute the general rules from specific rules.

The second section describes connectionist and symbolic (propositional and first-order) inductive methods for learning rules from examples. This section also compares the knowledge representation as propositional rules and more expressive first order rules. The preceding part of this section gives a detailed introduction to *rule extraction from ANNs*, and also surveys other work that has been done in this area. The later part of this section gives a brief overview of symbolic propositional rule learning techniques including learning *decision trees* and learning *logical concept definitions from examples.* The last part of this section introduces *first-order inductive learning* techniques. The material on symbolic rule learning is relevant since the GYAN methodology presented in the following chapters represents rules as a subset of first-order logic and forms the basis for a comparison to a first-order inductive learner.

2.1 Inductive learning

Machine learning (ML) is a branch of artificial intelligence concerned with the induction of knowledge or refinement of knowledge. The problem of inducing general concepts from specific training examples, by identifying features that empirically distinguish positive from negative examples, is central to machine learning. Over the last decade, machine learning has evolved from laboratory demonstration to the point where it has significant commercial value. Machine learning algorithms have been successfully applied to numerous difficult problems of practical interest [Mitchell, 1997], such as detecting credit card fraud by analyzing past transactions, steering a vehicle driven autonomously on public highways, and forecasting the most economical load distribution in a power system.

Inductive learning[1] is one of the main topics of study for researchers in machine learning (artificial intelligence), data-mining, statistics, cognitive science and related fields. The basic problem studied in inductive learning is to acquire a good representation of the desired function from given examples, *i.e.* a systematic relationship between inputs and outputs.

Research into inductive learning systems has been growing rapidly in three historically distinct areas: *computational learning theory* (which has undergone a renaissance in the last few years), *symbolic machine learning* (which has become a dominant influence in the field of artificial intelligence), and *connectionist/neural learning* (which has also seen enormous growth).

Computational learning theory provides answers to questions such as 'Under what conditions is successful learning possible and impossible?' and 'Under

[1]There is another field of study called *analytical or explanation based learning* [Dejong and Mooney, 1986], which states that learning requires *a priori* knowledge with few training examples (or no training examples), and only occurs at the periphery of this knowledge. In other words, learning in analytical systems is to improve the already known concepts.

what conditions is a particular learning algorithm assured of learning successfully?' [Mitchell, 1997]. Computational learning theory assists to identify techniques that can be applied into practical applications by providing theoretical analysis of the sheer size and complexity of the data. For example, this theory seeks to answer questions such as *sample complexity* (how many training examples are needed for a learning algorithm to converge to a successful hypothesis?), *computational complexity* (how much computational effort is needed for a learning algorithm to converge to a successful hypothesis?), and *error bound* (how many training examples are allowed to misclassify before the learning algorithm arrives at a successful hypothesis?) [Anthony and Biggs, 1995; Mitchell, 1997]. Although general answers to all these questions are not yet known, a broad range of learning models and learning algorithms deal with these questions. A useful collection of results under the 'probabilistic approximately correct' (PAC) model can be found in Blumer *et al.* [1989] and Anthony [1997]. A good overview of results in the area of 'query learning' is the paper by Angluin [1992].

Symbolic machine learning approaches emphasis the learning of heuristic, deterministic, and deductive models. These generate expressions (usually propositional or first-order logic) that are learned over the given attributes. Alternatively, connectionist/neural learning approaches emphasis learning of near-optimal (sometimes heuristic) stochastic models. These learn through incremental changes of weights in a network consisting of elementary units called neurons. Lallement and Alexandre [1997] gives a brief list of the capabilities[2] usually attributed to connectionist and symbolic models (see Table 2.1).

There are three basic types of inductive learning systems accomplishing

[2]Learning performance of connectionist models degrades while scaling up to large problems. For example, learning time in backpropagation neural networks often grows with the cube of the number of weights in the network [Collier and Waugh, 1994]. Learning time in decision trees is more or less linear in the product of the number of training examples and the number of attributes [Quinlan, 1986].

	Connectionist AI	Symbolic AI
Learning	Easy	Difficult
Generalization	Easy	Difficult
Noise tolerance	Good	Weak
Explanation	None	Good
	output - weights and bias	human readable
Data structure representation	Difficult	Easy
Performance degradation	Slow	Fast
Algorithmic	Parallel	Sequential
Scaling up to large problems	Difficult	Easy

Table 2.1: Some features of typical connectionist and symbolic models [Lallement and Alexandre, 1997]

supervised, unsupervised and reinforcement learning tasks. Supervised inductive learning task can be defined as determining the procedure for correctly assigning new unseen examples to target class(es), given the description of a set of examples each labeled as belonging to a particular class. In unsupervised learning, the training examples consist of features only, a target label(s) is not included. In reinforcement learning [Sutton and Barto, 1998], the state of a learning agent is observed and a set of actions is performed to alter this state. The task is to learn a control strategy for choosing actions that achieve the agent's goals. However, in this case the learner is not given an explicit output corresponding to a particular set of inputs. Instead it is periodically given performance indicators, *i.e.* a scalar *reinforcement signal.* The reinforcement learning task falls between the supervised and unsupervised learning tasks. The GYAN methodology developed in this thesis belongs to a class of supervised inductive learning algorithms predicting discrete-valued outputs.

A supervised inductive learning problem is often formulated as the problem of reconstructing the function f that maps domain X to range Y (i.e. it takes X as input and outputs Y) for a given set of examples, each example of f is a pair (x_1, y_1) where $x_1 \in X, y_1 \in Y$ and $y_1 = f(x_1)$. In machine learning terminology, the domain values $x_1, x_2, ..$ are called attributes, and the range values $y_1, y_2, ..$ are called target classes. Given a set of examples of f, the task

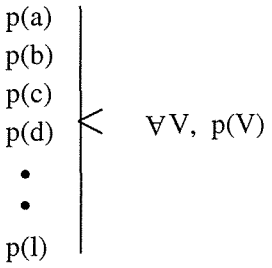
of inductive learning (generalization) is to return a function or a hypothesis H (description) such that H approximates f as closely as possible [Shavlik and Dietterich, 1990]. In an ideal inductive learning problem, an induced hypothesis H will agree with the classification of all the concept instances (training examples X). In practice, however, data given to the learner contains various kinds of errors, noise, etc., resulting in a hypothesis that does not completely agree with training examples.

The task of generalization is quite common in everyday life, and is fundamental in the process of human learning. Human learning of a new concept is possible after seeing a few positive (and negative) examples only. Michalski [1983] defines inductive generalization as the task of building a general description (hypothesis) from a set of examples such that the description can be used to obtain the prediction on new data. Many *generalization* and *reformulation* inference rules are required to induce an efficient description from pre-classified examples. A *generalization rule* is the transformation of a description (S) into a more general description (G) such that $G \models S$, one that tautologically implies the initial description [Michalski, 1983]. A *reformulation rule* is the transformation of a description into a logically equivalent description. Michalski [1983] suggests that a reformulation rule can be viewed as a special case of a generalization rule.

Many classical generalization rules such as *'dropping antecedents or conditions', 'turning constants into variables', 'turning conjunction into disjunction, 'climbing the generalization tree', 'extending the quantification domain', 'inductive resolution' and 'θ-subsumption'*, are extensively used in machine learning to implicate a *specific to general relationship.* Some of the generalization/reformulation rules that are further used in this thesis are:

1. **Turning constants into variables:** This rule is often used for generalization in inductive learning methods [Michalski and Chilausky, 1980]. If a number of descriptions with different constants are observed for a

predicate[3] or a formula, this rule generalizes these observations into a



generic predicate or formula. For example if a unary predicate (p) holds for various constants $a, b, ..l$ then the predicate p can be generalized to hold every value of a variable V with V being a constant a or a constant b, etc.

2. **θ-subsumption:** The θ-subsumption rule forms the basis of constructing and evaluating the hypothesis space in many first-order logic learners for concept descriptions. In this thesis, it plays an important role. The concept of general clauses generated using θ-subsumption provides the basis for mapping rules from a propositional representation to a representation using predicates.

R. J. Popplestone [1970] first introduced the idea that generalization of literals exists and is useful for induction. Plotkin [1970; 1971] then rigorously analyzed the notion of generalization to automate the process of inductive inference. He examined the properties of *first-order clauses under subsumption* and developed an algorithm for computing the *least general generalization* of a set of clauses.

Definition 1: (θ-subsumption) *A clause C θ-subsumes ($\preceq$) a clause D, if there exists a substitution θ such that $C\theta \subseteq D$. C is known as the least general generalization (lgg) of D (and D is specialization of C) under θ-subsumption if $C \preceq D$ and, for every other E such that $E\theta \subseteq D$,*

[3]The convention from logic programming and Prolog is followed, lower case names are used for predicates, functions, and constants, whereas variable symbols always begin with an upper-case letter (a more detailed format is given in [Lloyd, 1987; Wrobel, 1996]).

it is also the case that $E\theta \subseteq C$.

The definition is extendible to calculate the least general generalization of a set of clauses. The clause C is the lgg of a set of clauses S if C is the generalization of each clause in S, and also a least general generalization [Bergadano and Gunetti, 1995, pp. 35].

For example, $p(X, f(Y))$ is the least general generalization of $p(a, f(a))$ and $p(b, f(c))$. If the latter two literals form part of some data to be explained, then $p(X, f(Y))$ is the least general inductive hypothesis that explains them. To understand in detail consider the following clauses:

$C_1 : fish(canary) \Leftarrow has\_gills(canary)$

$C_2 : fish(salomons) \Leftarrow has\_gills(salomons)$

On applying the *'turning constants into variables rule'* and the *'dropping antecedents or conditions rule'* (*i.e.* the condition $fish(X)$ is dropped from the given clause) to the above clauses, the induced generalization will be: C_3 : $has\_gills(X)$, 'everything has gills'. From Definition 1, it can be observed that C_3 is a generalization of C_1 and C_2, but not a lgg. Based on every-day's knowledge, it can be observed that C_3 is an over-generalized clause and is clearly wrong. An alternative solution would be a generality relation between clauses according to the least general generalization under θ-subsumption. The lgg solution leads to $C_4 : fish(X) \Leftarrow has\_gills(X)$.

Muggleton and Deraedt [1994, pp.643] discuss several properties of θ-subsumption. Some of the properties are:

Infinite descending chains. Suppose, the following infinite descending chain exists:

$q(X_1, X_2) \Leftarrow p(X_1, X_2)$

$q(X_1, X_3) \Leftarrow p(X_1, X_2), p(X_2, X_3)$

$q(X_1, X_4) \Leftarrow p(X_1, X_2), p(X_2, X_3), p(X_3, X_4)$ ...

This chain is bounded under θ-subsumption by $C_5 : q(X,Y) \Leftarrow p(X,Y)$, that is incorrect (the correct generalization should be $C_6 : q(X,Y) \Leftarrow p(X,Y)$ and $q(X,Y) \Leftarrow p(X,Z), q(Z,Y)$). As illustrated, the *incompleteness of θ-subsumption* seems to be due to *self-recursive clauses.* Consequently, θ-subsumption is *not* enough to handle recursive clauses.

Implication. If $C \preceq D$ then $C \models D$ (D is observed by resolving C). But the opposite is not always true *i.e.* if $C \models D$ then sometimes $C \not\preceq D$. In other words, if an example is θ-subsumed by a hypothesis, it is also implicated by it, but if it is not θ-subsumed, it may still be implicated. For example, based on the two clauses C_5 and C_6 in the above example it can be said that $C_5 \models C_6$ but $C_5 \not\preceq C_6$, there is no substitution such that $C_5\theta \subseteq C_6$.

Plotkin [1971] has stated that *θ-subsumption is weaker than implication.* θ-subsumption is *reflexive* ($\Gamma \models \Gamma$), *transitive* ($\Gamma_1 \models \Gamma_2$, $\Gamma_2 \models \Gamma_3$ implies $\Gamma_1 \models \Gamma_3$) and is a proper subset of implication (or incomplete with respect to implication) [de Mantaras and Armengol, 1998, pp.105]. Gottlob [1987] has shown that if self-recursive clauses are not allowed in learning then *θ-subsumption is complete*[4] with respect to implication.

This property of θ-subsumption between clauses makes it *decidable*[5] *and easy to compute* [Bergadano and Gunetti, 1995, pp.35]. Wrobel [1996, pp.165] points out that θ-subsumption is a subset operation ($C\theta \subseteq D$) that may be preceded by substitutions on the more general clause. Since algorithms for computing a unifying substitution for individual liter-

[4] A relationship is considered complete if all the examples entailed by this are covered by a hypothesis.

[5] A relationship is decidable if a proposed solution by this relationship can be reliably checked whether it actually is a solution. If it is not, the computation necessary for checking the solution may not terminate. First order logical implication relationship ($\models$) is not decidable [Russell and Norvig, 1995]. If terms are considered to be function free in a representative first order language (only constants and variables are used in terms), then implication is decidable [Wrobel, 1996].

als exist [Lloyd, 1987], clearly this relationship is decidable, conversely to the fact that the implication between two Horn clauses is undecidable [Marcinkowski and Pacholski, 1992]. Garey and Johnson [1979, pp.264] have shown that checking θ-subsumption (*i.e.* matching all combinations of literals) is an NP-complete problem. The least general generalization of two clauses under θ-subsumption can *always be found* for Horn clauses with a common predicate symbol and having the same sign, but there is not always a least generalization under implication of two Horn clauses [Bergadano and Gunetti, 1995, pp.35].

Equivalence and Reduction. Two clauses equivalent under θ-subsumption are also logically equivalent (implication), an inductive learner generates at most one clause of each equivalence with a few redundant literals [Muggleton and Deraedt, 1994, pp.643]. Suppose, the following two clauses exist, $C_7 : q(X) \Leftarrow p(1, X)$ and $C_8 : q(X) \Leftarrow p(2, X) \wedge p(3, X)$ that result in the lgg clause, $C_9 : q(X) \Leftarrow p(V, X) \wedge p(W, X)$ under θ-subsumptions of {{V/1,W/1},{V/2,W/3}}. In other words C_7 and C_8 are equivalent with these θ-subsumptions. Plotkin [1971, pp.112] showed that there is a unique representative or a reduce clause (up to variable naming) of each equivalence clause. The reduced clause R of a clause C is a minimal subset of literals (excluding redundant literals l) such that C-{l} $\equiv$ R under θ-subsumption [Bergadano and Gunetti, 1995, pp.36]. A set of literals has a unique lgg but several lgg can exist for a set of clauses [de Mantaras and Armengol, 1998, pp.107].

3. **Counting arguments:** Constructive generalization rules generate inductive assertions during learning that use descriptors, originally not present in the given examples. There are many ways to compute new descriptors (predicates) during the generalization process. Michalski uses the *count quantified variables rule* (CQ) and the *count arguments of a predicate rule* (CA) to generate new descriptors in the general methodology, STAR [1983], which learns structural descriptions from examples.

For example, if a concept descriptor is in the form of $\exists V_1, V_2, .., V_l \cdot p(V_1, V_2, .., V_k)$, then the CQ rule generates descriptors $\#V\_cond$, representing the number of V_i that satisfy some condition *cond*. For example, if the descriptor is a predicate with several arguments, $p(V_1, V_2, ..)$, the CA rule generates new descriptors $\#V\_cond$, by measuring the number of arguments in the predicate that satisfy some condition *cond*.

4. **Term-rewriting:** This is a reformulation rule to transform compound terms in elementary terms. An *elementary term* (e-term) is the same as a term in predicate calculus, i.e., a constant, a variable, or a function symbol. A *compound term* is a composite of elementary terms. The composite of e-terms is defined as the *internal conjunction* ($\wedge$) or *internal disjunction* ($\vee$) of e-terms. A compound term, in which arguments are composite, can be transformed (expanded) into a composite of elementary terms. Let p be an n-ary predicate, whose first argument is a compound term consisting of t_1 and t_2, and the $n-1$ arguments are represented by a list A. The rules to perform such transformation are:

$$p(t_1 \vee t_2, A) \leftrightarrow p(t_1, A) \vee p(t_2, A)$$

$$p(t_1 \wedge t_2, A) \leftrightarrow p(t_1, A) \wedge p(t_2, A)$$

It is out of the scope of this thesis to cover all possible generalization rules, interested readers can refer to [Michalski, 1983] for a detailed description. Only some of the generalization/reformulation rules that are utilized in the thesis to form the general rules from examples, are described in this section.

2.2 Learning a set of rules

Inductive learning can be seen as the process of transforming source data into organized and usable knowledge. Inductive learners generate new general rules (concepts) on the basis of a collection of instances, and predict the value of dependent variable(s) from known attributes. It is now well accepted that the process of learning concepts from examples can be viewed as a search in a

hypothesis space for a concept definition(s) that explains all examples including both positive and negative. The search space is the set of states (or conceptual locations) through which a problem solver may conceivably pass in seeking the solution to a problem [Mitchell, 1982]. Every learning algorithm has an *inductive bias* that determines the representation, construction, evaluation and modification of hypothesis space and hence, the learned model [Mitchell, 1980]. There are two aspects of the *inductive bias* of an algorithm [Hilario, 1997]: its *representational bias*, sometimes called *restricted hypothesis space bias*, that refers to the choice of language used to describe hypothesis or generalization; and its *search bias*, sometimes called *preference bias*, that determines the order of traversal of the spaces to select a hypothesis that approximates the target concept. According to Russell [1996], there are four major ways in which a hypothesis is represented:

Artificial neural network: This is a real-valued/discrete-valued/vector-valued, linear/nonlinear function represented by the parameterized network of simple computing elements. This representation describes objects by a vector of predefined features. These input attributes may be highly correlated or independent of one another. The output of a learning procedure is represented by a vector of weight parameters according to the architecture of the network. The acquired knowledge may be transformed into a more comprehensible representation (such as prior specified attributes in a disjunction or conjunction or rules with variables) by further processing.

Attribute-based representation: This is a boolean or multi-valued function that provides a decision based on the logical (propositional) combination of input attributes. This representation describes objects as the fixed collection of attributes, each of them taking a value from the corresponding pre-specified set of values. The decision tree representation falls into this category. Some ANNs and belief networks can also be included in this category.

First-order logic: This is an expressive formal language that includes quantification, relations, recursion and iteration. This representation describes objects in terms of their components and relations among components.

Probabilistic functions: This type of representation returns a probability distribution over the possible output values for any given input, and is suitable for problems that require answers with some probability. Belief networks and learners using fuzzy logic as representation fall into this category.

The selection of a knowledge representation language is very important in inductive learning since it defines the hypothesis space: what knowledge can be expressed and, therefore, what knowledge can be learned. There are a variety of inductive learning algorithms that learn a target function based on the above described representations. The efficiency and tractability of a learning algorithm is affected by the choice of representation [Garey and Johnson, 1979]. There is a fundamental trade-off between *expressiveness (of the representation language)* and *efficiency (the computational time and learnability)* for inferencing rules from examples using various inductive learning algorithms.

Rule representation language: propositional vs predicate logic

The target function (or the transformed learned knowledge) is usually represented as a set of propositional (if-then) rules or a set of first order logic (predicate) rules. There is a considerable difference in presentation of rules using both logics. Propositional logic is a mathematical model (or algebra) for reasoning about the truth of logical expressions (propositions) [Edmond, 1992]. Propositional logic is restricted in the sense that it deals only with truth values of complete statements and does not consider the general relationship or dependencies between objects [Grassmann and Tremblay, 1998]. When information in a domain is presented using propositional logic, it usually results in a large data base containing 'all the known facts' about the relevant knowl-

edge in the domain.

The fundamental problem for the representation of knowledge using propositional logic lies in a combinatorial explosion - increase in the number of deductions possible from a set of input propositions [Ramsay, 1988]. This number tends to grow exponentially with the number of propositions, thereby making this type of representation impractical for all but the simplest data bases [Firebaugh, 1989]. Concepts for general class variables, quantifiers, and functional relationships are lacking in a propositional rule-base.

Consider the following boolean propositions; A: (Adam is tall), B: (Beth is tall), C: (Carl is tall),...., and Z: (Zeke is tall). A different proposition for each person is needed; each of these propositions is either true or false. By the use of predicate calculus, the same set of truth values (boolean propositions) can be captured in a single predicate *i.e.* these propositions can be represented by the unary predicate, $tall(X)$, where a person's name is represented as the variable X. The predicate is true whenever the person X is tall, and is false otherwise. For example, $tall(adam)$ is true if proposition A above is true. Predicate logic (invented by Gottlieb Frege) combines propositional logic's ability with the syllogistic ability to delve into the internal structure of the boolean proposition [Bundy, 1980], and to a method for deciding the correctness of non-boolean arguments.

First order predicate calculus[6] is an extension and generalization of propositional logic which provides a more powerful formalism to make logical inferences [Russell and Norvig, 1995]. Predicate logic extends the rules of propositional calculus by introducing predicates (describing properties or relations of objects in a certain domain), quantification (variables ranging over arbitrary domains), and inference rules for quantified predicates. The notions of *seman-*

[6]Also called first order logic or just simply predicate logic, and sometimes abbreviated as FOL or FOPC [Russell and Norvig, 1995, pp.185].

tic equivalence, tautology, contradiction and satisfiability can be transferred directly from propositional logic to predicate logic [Lloyd, 1987]. All valid formulas of propositional logic are valid in predicate logic [Ramsay, 1988]. The only difference lies in the notion, *'binding of variables'*, which allows a substitution of variables by elements in the domain. Rules of inference like resolution and para-modulation apply only to predicate logic (in clausal form) [Thomas, 1989]. This capability has made predicate logic an essential technique in specific fields of AI such as automatic theorem proving, etc. First-order predicate logic offers a powerful, uniform, and elegant way to describe and formalize even relatively complex domains.

The next sections explore several ways to learn a set of rules such as: (1) learn an ANN, then translate the architecture and weights into an equivalent set of rules; (2) learn a decision tree, then translate the tree into an equivalent set of rules; or (3) learn the rule sets that contain variables.

2.2.1 Rule extraction from ANNs

ANNs are a powerful general purpose tools applied to many machine learning tasks. The ANN learning method provides a robust and non-linear approach to approximating the target function for classification (discrete-valued), regression (continuous-valued) and clustering problems [Mitchell, 1997]. ANNs have been successfully applied to many practical problems [Shavlik *et al.*, 1991] such as interpretation of complex real-world remote sensing data [Hammadi and Korczak, 1995], recognition of handwritten characters [Lecun *et al.*, 1989], spoken words [Lang *et al.*, 1990], and faces [Cottrell, 1990], forecasting of an economical generating schedule for a power system [Nayak and Sharma, 1999], modeling complex environmental data, force predictions in mills, machine intelligence in a mass transit railway system and self-calibration of a space robot [Dillon *et al.*, 1997]. The basic problem solved by ANNs is the *inductive acquisition* of concepts from examples. The ability to learn and generalize from data, which mimics the human capability to learn from experience,

makes ANNs useful in automating the process of learning rule sets.

Several empirical studies [Atlas *et al.*, 1989; Weiss and Kapouleas, 1989; Miller *et al.*, 1990; Shavlik *et al.*, 1991; Quinlan, 1993b] have shown that there are some problem domains (consisting of numerical-valued features or where all the input features are relevant to the classification) in which ANNs provide superior generalization accuracy as compared to competitive symbolic machine learning methods. The reason is that ANNs induce hypotheses that generalize better than those of competitive symbolic learning algorithms, and are capable of representing nonlinear, nonmonotonic continuous or linear discriminant functions [Weiss and Kulikowski, 1991; Craven and Shavlik, 1997; Sun and Bookman, 1995]. Unlike most symbolic learning systems which explicitly search for a simple hypothesis, ANN systems simply search for a correct hypothesis which fits into a user specified network [Shavlik *et al.*, 1991]. Quinlan [1993b] has observed that the backpropagation method of neural learning explores a larger hypothesis space, thus taking longer to find a suitable hypothesis, but a better fit is possible as compared to the decision tree method of symbolic learning. Also, ANNs have parallel processing capability, learning and feature extraction ability, robustness to small perturbations, nonlinear modeling capability and can be initialized with *a prior* knowledge [Bechtel and Abrahamsen, 1991; Rojas, 1996; Dorffner, 1997].

A recognized shortcoming of ANNs is the inability to explain the decision process in a comprehensive form by which a trained network arrives at a specific conclusion. Understanding a trained ANN is desirable for many reasons. For a medical diagnosis, airline or power station security system, it is imperative that the system's users should be able to validate output of the trained ANN under all possible input conditions.

A number of *scientific visualization* techniques have been utilized to understand different aspects of ANNs such as: (1) visualizing nodes and weighted

connections in the trained network [Hinton, 1986]; (2) visualizing the decision boundaries formed by nodes in the network [Pratt *et al.*, 1991; Maire, 1999; Melnik and Pollack, 1999]; and (3) visualizing the forward and backward propagation of activation signals through the network [Craven and Shavlik, 1992]. Most of these visualization methods are incapable of providing a complete description of the concepts learned by the trained ANN. Also, several of these methods only work well for small networks [Craven and Shavlik, 1992].

A number of researchers have employed *statistical techniques* to characterize the activity of hidden nodes in the trained ANN such as: (1) the hierarchical clustering of hidden-node activations [Hanson and Burr, 1990; Rumelhart, 1990]; (2) contribution analysis [Sanger, 1989]; and (3) weight pattern analysis [Gorman and Sejnowski, 1988]. Like visualization techniques, the hidden node analysis methods are incapable of providing a complete description of the concepts learned by the network.

One of the most promising approaches, to overcome the incomprehensibility of the knowledge acquired by a trained network, is to translate the stored knowledge (connection weights) into symbolic rules. Rule extraction from ANNs can help to explain their behavior and also facilitates the transfer of learning (from ANNs to expert system by automating the knowledge bases). The exercise of rule-extraction from ANNs is important due to: (1) in real life situations, systems that declare the learned knowledge explicitly are adopted more freely (such as symbolic machine learning systems); (2) the rule base generated from the trained ANN is sometimes sufficient for accurate modeling of the given domain [Sestito and Dillon, 1994]; and (3) in some cases the ability to explain how a solution is arrived at is essential in practical systems (for example controlling power regulation in a power system). The importance of rule extraction from ANNs can be illustrated in many aspects [Andrews *et al.*, 1995]. Some of them are:

- *Provision of a user explanation capability.* Symbolic inductive learning

techniques represent the learnt knowledge declaratively, often in the form of 'if-then' rules. However ANNs do not have such capability. Rule-extraction techniques extract the knowledge acquired by an ANN so that user explanation is possible. Rule extraction from ANNs significantly enhances the capability of an ANN to explore data for the benefit of a user. It allows to explore and induce data for scientific theories [Andrews *et al.*, 1995]. Neural networks identify dependencies and relationships in data and predict future behavior, and rule extraction techniques provide means to understand the capability of ANNs.

- *Data-mining and knowledge discovery.* The main focus of the data-mining and knowledge discovery enterprise is to identify valid, novel, potentially useful, and ultimately understandable patterns, and thus to discover hidden information from a large collection of data without a previously formulated hypothesis [Fayyad, 1996]. This goal can be achieved by applying ANNs to inductively construct the data at hand. An advantage of using ANNs is that they can incorporate background knowledge during learning [Fu, 1995; Towell and Shavlik, 1993], which can be invaluable for knowledge discovery tasks. Initializing an ANN with a prior symbolic knowledge helps to determine the representation bias (such as network architecture) and the search bias (such as initializing link weights and determining the initial state from which to conduct search during learning) of the ANN, and thus helping enormously to determine the learned model from data [Hilario, 1997].

  Although ANNs have been successfully applied to a wide range of supervised and unsupervised learning problems, they have not frequently been applied in data-mining [Craven and Shavlik, 1997]. A fundamental consideration behind this setback is the poor comprehensibility of learned models. Rule extraction from ANNs addresses the comprehensibility issue and makes ANNs suitable for data mining. In case of limited or incomplete data, the generalization capacity of a trained ANN may be poor. In such situations, a set of symbolic rules expressing the knowl-

edge embedded within the trained ANN may be able to help users to anticipate or predict a set of circumstances under which generalization failure can occur [Andrews *et al.*, 1995]. In other words, Rule extraction techniques may help the user to supplement data by identifying regions in input space which are represented insufficiently during training [Andrews *et al.*, 1995]. Thus ANNs result in allowing better generalization for data-mining. Integration of ANN's knowledge as rules with an inference engine provides user interface.

- *Knowledge acquisition for expert systems.* In the late 1970s, knowledge-based systems became popular in computer science as a result of the seminal results obtained by the DENDRAL system [Linsay *et al.*, 1980] to elucidate chemical structures and the MYCIN system to diagnosis infectious diseases [Buchanan and Shortliff, 1984]. In the early 1980s, expert systems started to become commercially available and the development of knowledge engineering tools to accumulate, apply, validate and maintain knowledge (in form of facts and/or rules) that powers such applications, became a potential research field. A decade after these initial commercial ventures began, thousands of knowledge based systems have been developed and deployed in industrial and commercial usage. However, the exercise of knowledge acquisition (generating knowledge bases) continues to challenge expert system builders [Boose, 1991]. One of the promising approaches is to automate the rule induction process from low-level data rather than emulating the behavior of experts considering that knowledge of an expert system is often presented in rules [Datta, 1993]. Empirical inductive learning techniques have been successfully applied to extract high-performance knowledge structures (rules) from cases supplied by experts or from initial low-level data (which may also contain noise or missing values) [Muggleton, 1990]. They have also been employed to construct an expert system by transforming the knowledge gained by an ANN into symbolic rules and/or facts [Nayak *et al.*, 1999].

Andrews, Diederich and Tickle [1995] developed an overall taxonomy for

categorizing techniques that extract rules from trained ANNs. The taxonomy recommends five primary criteria: (1) the representation language of the extracted rules; (2) the translucency of the view of the underlying network architecture; (3) the quality of extracted rules; (4) the computational complexity of the rule-extraction technique; and (5) the generality of the rule-extraction technique according to the need of specialized training regimes.

The first classification criterion focuses on the representation language of the output of a rule-extraction technique. The extracted rules are expressed in various forms such as: (1) *propositional/boolean logic* (i.e. if....then....else, m-of-n inference rule); (2) *non-conventional* approaches (i.e. rules with confidence factors or 'fuzzy logic' rules); and (3) *first-order predicate logic* (i.e. rules with quantifiers and variables). Several algorithms have been developed to extract rules from trained ANNs in the past few years. Most of the algorithms have focused on representing the extracted knowledge in the form of propositional logic. The GYAN methodology presented in the following chapter extracts rules in the form of *restricted first-order predicate logic.*

The second classification criterion reflects on the relationship between the extracted rules and the internal architecture of the underlying ANN. This criterion extends the classification schema used by Craven and Shavlik [1994]. This criterion comprises two basic rule extraction categories: *decompositional (local)* and *pedagogical (global)* and; a third category labeled as *eclectic* which combines elements of the previous two basic categories. GYAN applies both the decompositional and pedagogical approaches to extract restricted first order logic rules. The rule-extraction methods that are discussed in the following sections, express rules in propositional language and differ along the second classification dimension.

Decompositional rule extraction techniques representing propositional rules

The distinct feature of *decompositional* approaches is the *high level of granularity* in the extracted rules. The decompositional rule extraction techniques extract rules by decomposing a multi layer network into a collection of single layer networks or nodes. The aim is to extract rules at the level of each individual hidden and output node, and then aggregate to form the composite rule base that describes the network as a whole. The rationale is that the function learnt by the trained network is easier to express in terms of intermediate concepts and in turn, the intermediate concepts are easier to express in terms of original attributes. It is generally believed that such a structuring of the learning domain leads to a more comprehensible description of the target concepts. A single layer network, describing a simple (maybe linear) relationship between input features and an output category is easier to understand.

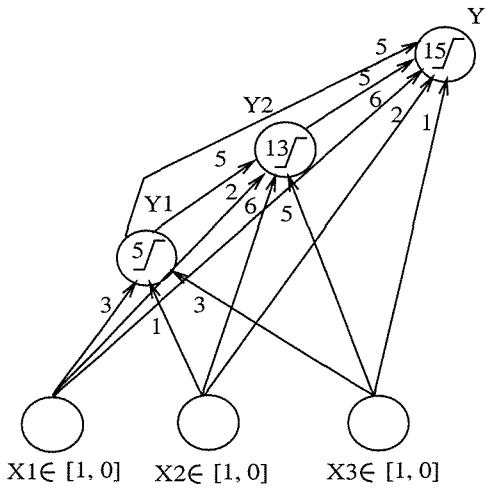


Figure 2.1: A simple cascade network with boolean input and output space.

The three single-layer networks in Figure 2.2 are obtained when the network in Figure 2.1 is decomposed. Assume that the output and inputs to the network are boolean, and the non-input nodes employ a threshold function to compute the activation; that is, a node produces an output of one only if the sum of its

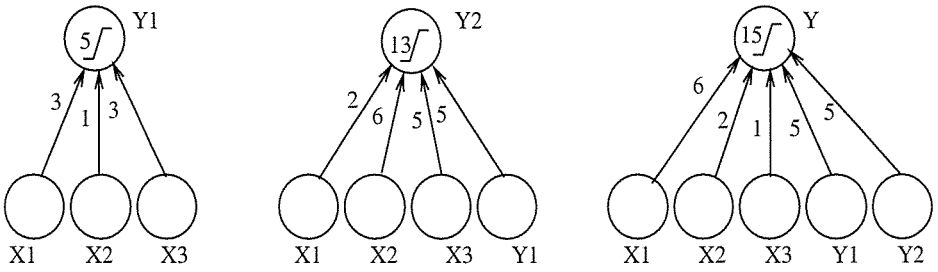


Figure 2.2: The decomposed networks. All the input nodes including hidden nodes (as input to the next node) are assumed to have boolean inputs.

weighted inputs exceeds a certain threshold.

$$a_y = \begin{cases} 1 & \text{if } \sum a_i * w_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

Based on the assumptions, rules for the composite network will be:

$$Y := Y_1 \wedge Y_2 \wedge X_1$$

$$Y_2 := X_2 \wedge X_3 \wedge Y_1$$

$$Y_1 := X_1 \wedge X_3$$

The earliest decompositional rule-extraction method is the *KT* algorithm developed by Fu [1994]. The *KT* algorithm explores a very basic principle of ANNs; that is, if the sum of its weighted inputs exceeds a certain threshold then the neuron fires. The overall search of the algorithm iterating over every non-input node in the trained ANN is exponential in the number of inputs to the network unless some heuristics are employed [Fu, 1994]. The same idea is incorporated in the SUBSET algorithm, developed by Towell and Shavlik [1993]. Towell and Shavlik further noticed that the rule sets discovered by the SUBSET algorithm often contain an M-of-N style of representation. Addressing both the representation and combinatorial problems inherent to SUBSET and *KT* algorithm, they developed the *MofN* algorithm [Towell and Shavlik, 1993].

The idea underlying the *MofN* algorithm is that groups of similar antecedents have a unique importance rather than the individual antecedents in isolation. The *MofN* decompositional rule-extraction method extracts sym-

bolic rules in the form of, if (M of the following N antecedents are true) then (the target value). However in many cases, the extracted rules take the form of a linear inequality involving multiple numeric quantities providing too much details, such as if (1 of $(X_1, X_2, X_3) \wedge$ 2 of (X_2, X_5, X_7)) then (the hidden node will fire). In fact *MofN* rules provide a precise mathematical description rather than the nearest symbolic interpretation of a node behavior, and may be difficult to interpret and to use in reasoning. The *MofN* algorithm imposes some restrictions on the underlying networks. *MofN* requires the underlying ANNs to be trained with a special procedure so that networks with clustered weights are produced after training such as: KBANN in which the underlying network is initialized by a domain theory [Towell and Shavlik, 1993]; or to use soft weight-sharing during learning to obtain a network state with clustered weights [Craven and Shavlik, 1993]. The *MofN* method relies substantially on access to a fixed set of training patterns by applying heuristic elimination (querying the network) to prune the clusters. The *KT*, SUBSET and *MofN* rule-extraction algorithms, similar to other decompositional rule-extraction methods, assume that a non-input node is either maximally active (activation near one) or inactive (activation near zero). These methods require a hard-limiting threshold function or an approximated logistic activation function for a non-input node to extract rules.

McMillan *et al.* [1991] developed the technique called *RuleNet & The Connectionist Scientist Game* that incorporates the ANN training with a decompositional approach for rule-extraction, and rule-refinement which allows to inject the extracted rules back into the network. The major shortcoming of this approach is the lack of generality due to the use of specialized neural training regime. Alexander [1994] utilized the basic idea of *RuleNet*, *i.e.* simple symbolic interpretation using *weight template* techniques, and applied to feed-forward multi-layer networks trained with backpropagation algorithm. *Weight templates* are produced as parameterized regions of the total weight space corresponding to a given m-of-n expressions. A *weight template* that best fits the

actual weights in the trained ANN is selected, and directly interpret in rules. This rule-extraction technique successfully extracts *MofN* rules for simple processing tasks, but the technique fail for multilayer networks when hidden units do not employ boolean activations. Tresp *et al.* [1993] focussed on the idea of utilizing the prior knowledge in the network's training and developed a technique for a probabilistic interpretation of the ANN architecture based on the Gaussian basis functions as classifiers.

Another significant development in decompositional rule extraction techniques is the *RULEX* method of Andrews and Geva [1994]. *RULEX* performs rule extraction by directly converting weight vectors to rules rather than search and test. The *RULEX* technique is exclusively developed to extract rules from the constrained error backpropagation (CEBP) network that performs function approximation and classification very similar to radial basis function networks (more detail in Chapter 3).

Sethi *et al.* [1994] proposed the backtracking tree search procedure to convert the weights of a neuron into a symbolic representation. The *backtracking tree* algorithm is based on a heuristic search in the depth-first tree which is formed by recursively extracting all essential prime implicants of the weight vector for each non-input neuron by using solution and boundary functions. The problem of this algorithm lies in the large number of nodes in the generated trees.

One additional approach which falls into this category is the *LAP* algorithm [Hayward *et al.*, 1996]. Assuming a sparse coded input space, the *LAP* algorithm (more detail in Chapter 3) forms a set of weight vectors, each vector corresponds to the weights of each sparse-coded attribute. The algorithm imposes an order to evaluate elements in each weight vector according to the magnitude of weights. A recursive procedure is employed to test the sum of the largest weights from each vector against the bias weight. This method

is guaranteed to extract rules with 100% fidelity for a perceptron but suffers from the curse of dimensionality. The overall search of the algorithm iterating over every non-input node in the trained ANN is exponential in the number of values that all the input attributes hold in the problem domain.

Taha *et al.* [1997] developed the *Full-RE* method that extracts all the possible embedded knowledge in the trained ANN. The *Full-RE* approach discretises the boundaries of input features to handle continuous input values. Given the discretisation boundaries of each input feature, *Full-RE* applies the *linear programming method* to minimize the weighted input vectors. Considering that extracting minimal rules from a feedforward network is a NP-hard problem, but a rule-extraction problem can be efficiently solved by identifying subclasses of networks and subclasses of rules [Golea, 1996], the authors developed a comprehensive technique to extract the key (most important) rules only.

A recent development is the *Neurorule* [Setiono, 1997a; Setiono and Liu, 1996] rule-extraction technique. The rules extracted by *Neurorule* are comparable to those generated by decision trees in terms of accuracy and comprehensibility. The distinct quality of this algorithm is to replace the continuous activations of hidden nodes by a small number of discrete ones using Chi2 [Liu and Tan, 1995]. This method only works for binary inputs. Data with continuous attributes needs to be discretises before training the network. An inherent problem introduced by discretising data is that each condition of a rule involving a continuous attribute determines an axis-parallel decision boundary [Setiono and Liu, 1997].

NeuroLinear [Setiono and Liu, 1997] improves *Neurorule* by allowing oblique hyperplanes to form the boundaries of decision regions resulted from continuous attributes. The oblique hyperplanes allow to form oblique (*i.e.* neither perpendicular nor parallel) decision boundaries instead of dividing the decision regions into many small rectangular regions like decision trees to approximate

a function [Setiono and Liu, 1997]. *NeuroLinear* extracts oblique decision rules from trained ANNs such as if (monthly salary - 1.5*monthly mortgage) > 0 then saving. The recently developed FERNN algorithm [Setiono and Leow, 1999] identifies the relevant hidden nodes of the trained ANN by C4.5 [Quinlan, 1986], instead of discretising hidden activation space. The relevant connections from input nodes to hidden nodes are identified according to the magnitude of their weights.

Pedagogical rule extraction techniques representing propositional rules

In a *pedagogical* approach, the trained ANN is treated as a *'black box'*. The rule extraction task is viewed as a *learning task* where the target concept is the function computed by the trained network and the input features are simply the network's input features. The objective is to extract a set of rules that characterizes the target concepts directly in terms of the inputs. [Andrews *et al.*, 1995]

One of the earliest pedagogical rule-extraction methods is the algorithm of Saito and Nakano [1988]. This method is a search based approach that explores the space of candidate rules created by altering the values of input and output nodes, and then tests the candidate rules against the network (treating it as a 'black box'). It deals with the combinatorics of the task (a large number of possible combinations of inputs) by limiting the number of antecedents that can be present in a rule. Even with these heuristics, the number of extracted rules for a relatively simple problem domain is exceedingly large.

Gallant [1993] developed a method which tests the rules against the network by propagating *activation intervals* through the network. Sometimes this algorithm fails to accept maximally general rules, as it is guaranteed to accept only rules that are valid and overly specific. The *validity interval analysis* (VIA) technique developed by Thrun [1995] is a generalized and more power-

ful version of this technique. The VIA technique specifies a maximum range for the activation value termed *'validity interval'* for each individual unit instead of propagating the accurate activation signals. Although the VIA method is better at detecting general rules than Gallant's algorithm, it may still fail to confirm maximally general rules [Craven and Shavlik, 1997].

Craven and Shavlik [1994] applied the PAC (probably approximately correct [Kearns and Vazirani, 1994]) algorithm, *'rule-extraction-as-learning'*, to extract rules from trained ANNs utilizing the 'pedagogical' approach. The basis of this algorithm is the repeated querying of two oracles *(Example and Subset)* and the formation of disjunctive normal form (DNF) expressions based on the decisions made by the ANN for the queried instances. However there is no guaranteed rate of convergence in reaching a satisfactory rule set. Pop *et al.* [1995] developed the *RuleNeg* pedagogical rule extraction technique based on PAC learning techniques. The *RuleNeg* algorithm utilizes the *Subset oracle* to justify assignment queries. The assignment query is an amended instance together with its original target class. The network's responses to the amended queries are analyzed and the DNF expressions are generated. Recently Hayward *et al.* [1997] pointed out the shortcomings of *RuleNeg* and indicated that functional dependencies between dimensions can lead to degenerative rule sets for *RuleNeg*. To improve the quality of rules obtained by *RuleNeg*, the *RuleVI* algorithm [Hayward *et al.*, 1997] utilizes the VIA method to allow *Subset* to ask partially specified queries to validate a proposed rule (more details in Chapter 3).

Fletcher and Hinde [1995] developed an algorithm to produce a correct boolean representations by analyzing truth tables. A truth table is generated by examining the network's response for each input pattern. This truth table is then transformed into a boolean function using an algorithmic implementation of Karnaugh maps. As the number of inputs grows, so does the length of the Boolean representation, and the algorithm fails to extract rules. One more

concern with this algorithm is exceedingly large size of the derived rule bases.

The TREPAN algorithm [Craven, 1996] is another significant development to extract a comprehensible hypothesis from a trained ANN. In this inductive learning task, the target concept is the function computed by the network, and the hypothesis produced by the learning algorithm is a *decision tree that approximates the trained network.* The TREPAN algorithm is limited to describing the real-valued predictions of models trained to perform regression tasks. Also it does not guarantee the extraction of a tree that exhibits a high level of fidelity to its underlying network [Craven, 1996]

Taha *et al.* [1996] proposed the pedagogical rule-extraction algorithm, BIO-RE, that extracts the rules from an ANN trained with binary inputs. The method generates a truth table by concatenating each input in feature space and the corresponding output decision made by the trained network. Finally, the corresponding boolean function is generated from the truth table represented in the binary rule-format. The algorithm developed by Castellanos *et al.* [1997] utilizes Gallant's method [1993] to extract rules from the trained ANN. This algorithm first uses sensitivity analysis [Viktor *et al.*, 1995] to obtain the most important variables for further prediction. To handle regression problems, the target output is divided into ranges and the network is trained for each output range. In both the algorithms, the generated rule sets suffer from the problem of poor comprehensibility.

Eclectic rule extraction techniques representing propositional rules

The *eclectic* approaches incorporate elements of both the *pedagogical* and *decompositional* rule extraction approaches, and are identified by the degree of translucency that they require of the network while extracting rules. This type of algorithm utilizes knowledge about the internal architecture and/or weight vector in trained ANNs to complement a symbolic learning algorithm to extract rules from ANNs.

The *Rule-extraction-as-learning* algorithm developed by Craven and Shavlik [1994] (mentioned in the previous section as a pedagogical rule-extraction method) can also be categorized as an eclectic rule-extraction method depending on the particular implementation used. The use of a function called *Subset*, to determine if the modified rule still agrees with the network, makes the method flexible. A decompositional procedure like the KT algorithm [Fu, 1994] or a pedagogical procedure like VIA [Thrun, 1995] can be used in a *Subset* oracle. The empirical results prove the method to be more efficient than conventional decompositional or pedagogical approaches. In addition to learning from training examples, their method exploits the property that the networks can be *queried.*

The recent LBSB [Yuanhui *et al.*, 1997] algorithm combines learning and searching techniques together to extract rules from a three-layer backpropagation ANN. The LBSB algorithm first utilizes a learning method which considers only the hidden and output layers in the network. This learning algorithm identifies regions in the activation space of hidden units so that all the vectors (activation of hidden units) in the identified valid regions generate activation of a given output larger than a certain threshold. The second phase searches for rules between the input and hidden layers, such that all instances covered by the rules generate the activation vectors at hidden units that fall in the valid regions. This method extracts rules with high fidelity but with poor comprehensibility.

Another recent example of eclectic approaches is the *DEDEC* methodology [Tickle, 1998] applicable to a broad class of multilayer feedforward ANNs trained by the backpropagation algorithm. This methodology identifies functional dependencies between inputs and outputs of an ANN by analyzing the architecture and weight vectors of the trained network. The *DEDEC* technique searches the solution space by ranking the ANN inputs according to

their relative share of contribution to the ANN output(s) and extracts rules. This technique needs refinement to provide more comprehensible rule sets as it is designed to extract symbolic rules from a set of individual cases [Tickle, 1998].

Rule-extraction: Discussion and suggestions

One of the problematic issues that arises in rule-extraction techniques that are cast as a search problem, is that the size of the hypothesis space for searching rules can be very large, which generally results in *computationally expensive* methods. For a problem domain with n binary features (values in {positive, negative, absent}), there are 3^n possible conjunctive rules that can represent the underlying problem. In other words, the search space grows exponentially with the number of input features and the values that they contain. Golea [1996] has proved that extracting minimal rules from a feedforward network is a *NP-hard* problem. Further Golea suggests that there are always some cases where a rule-extraction technique will fail, but a rule-extraction problem can be efficiently solved by identifying subclasses of networks and subclasses of rules.

Rule-extraction techniques that utilize a network's responses by repeatedly querying the network to extract rules (based on the PAC techniques), rely substantially on access to a set of training patterns and result in rules that are overly specific. Furthermore, Golea [1996] showed that the primary query based algorithm *Rule Extraction as Learning* [Craven and Shavlik, 1994] is not polynomial in the worst case, as it is based on Valiant's greedy method [1984] for learning DNF expressions. In other words, if a rule-extraction technique utilizes amended instances to query the network, the technique suffers from a large number of possible combinations of query instances (that are produced by amending each instance in the *example* oracle) to validate the proposed rules.

To address the computational complexity issue, a number of heuristics have been explored that limit the size of the solution space which is searched to generate the rules [Tickle *et al.*, To appear]. A direction taken by some rule extraction techniques is to *impose certain types of restrictions* on the underlying *network architecture* (such as *MofN*) or on the *training algorithm* (non-input units using a Heaviside activation function or a steep sigmoid, or initialization of networks with prior knowledge) or on the *input space* (only suitable for problems involving discrete-valued features). Although these requirements may simplify the rule-extraction process and improve rule-quality, the trade-off involved is that they reduce the fidelity of the extracted rules to their target networks and the generality of the rule extraction method.

An alternative approach to efficiently extract rules from the trained ANN is to reduce the size of the hypothesis space by removing superfluous nodes and links from the trained network. Heuristics are needed to guide the extraction process, which searches for a rule in the given ANN trained for a problem. An important point to guide the search is the *quality of attributes*. Some of the attributes may be irrelevant for the task at hand, as people tend to collect data along all the dimensions that they consider probably useful because of the lack of knowledge about an application domain. When a large number of attributes are involved in the problem domain, some relevant attributes may become redundant in the presence of other attributes.

As a result of data-preprocessing step (data cleaning and feature selection), only a specific selected subset of attributes are presented to an ANN. ANNs are able to determine relationship among attributes when a complete set of attributes is presented, demonstrated by many empirical studies [Widrow *et al.*, 1994]. A better option is to first learn an ANN on the complete set of attributes and then select a reasonable subset of attributes so that the chosen subset can adequately explain the target concept. There are many ways to select dependent/significant attributes in the trained network. Some of them

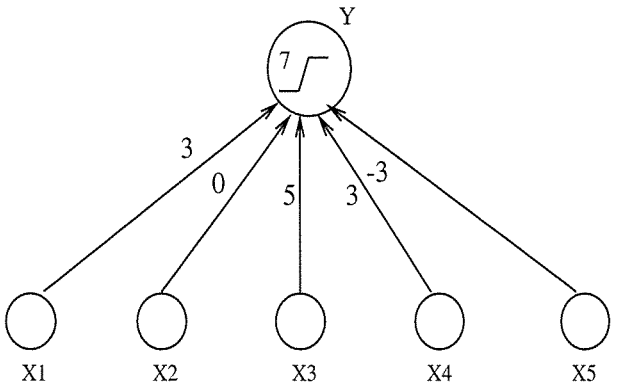


Figure 2.3: A simple one layer ANN with boolean input and output space, and a hard-limit threshold function

are: (1) eliminating irrelevant input nodes corresponding to attributes/values in the problem domain by *pruning* after the ANN learning process has been completed [Prechelt, 1995; Setiono, 1997a]; (2) ranking a network's inputs according to their *relative share of contribution* to the predicted output(s) based on the generated weight states in the trained network [Tickle, 1998]; and (3) utilizing *sensitivity analysis* [Viktor *et al.*, 1995] between the inputs and the predicted output(s) of a trained network or each weight in the network. If the sensitivity is large, the weight or input is considered important. It the sensitivity is small, then the influence of the weight or input is considered unimportant.

Reducing the number of attributes not only speeds up the extraction process, but also prevents the generation of an inferior rule set (due to the presence of many irrelevant attributes). This is mainly because most of the practical extraction algorithms are heuristic in nature (*i.e.* exploring a space of candidate rules to test the validity of the rule against the network), and they are often misled by the presence of many non-essential attributes. By using relevant attributes/values, extraction algorithms can in general improve their predictive accuracy, shorten the extraction period, and improve comprehensibility by simplifying the extracted rule set. For an example, refer to the single-layer network in Figure 2.3, input and output spaces are assumed to be boolean. Assuming that the output node employs a hard-limit threshold func-

tion to compute the activation; that is, the output of one is produced only if the sum of the weighted inputs exceeds a certain threshold $|\sum a_i * w_i > \theta|$. The weighted connections in the network are translated into the symbolic rules:

$$Y := X_1 \wedge X_3 \wedge X_4$$

$$Y := X_1 \wedge X_3 \wedge \neg X_5$$

$$Y := X_3 \wedge X_4 \wedge \neg X_5$$

The attribute X_2 does not appear in the extracted rules, so it can be removed from the network without effecting the network's performance. This indicates that connections with sufficiently low weights are deemed inconsequential and do not take part in symbolic rules. Removing this type of attribute can be seen as having a beneficial effect on simplifying the rule-base. Pruning of the trained network that has been done correctly, selects the minimum number of attributes needed to represent the data accurately, and helps to obtain a concise set of symbolic rules [Blassig, 1994; Prechelt, 1995; Setiono, 1997a].

Another issue in rule-extraction techniques is the *comprehensibility of extracted rule sets.* However, pruning and other restrictions imposed on network learning facilitate production of comprehensible rule sets from the trained ANN. Sometimes, it is very hard to understand the large number of propositional rules extracted by a rule-extraction technique, and some type of post-processing is required. An easier way is to map the hundreds of specific rules into a fewer general first-order logic rules or a subset language. The first-order logic rules allow to learn general rules as well as the internal relationships among the variables. The difference can be illustrated by the following example. Suppose the task is to learn the target concept *wife(X, Y)* defined over the pairs of people X and Y. Based on one positive example: (Name = mary, Married_to = john, Sex = female, Wife = true); a propositional rule learner will learn a specific rule, *If (Name = mary)* $\wedge$ *(Married_to = john)* $\wedge$ *(Sex = female) Then (Wife = true)*, and similarly many will be generated from other examples. The program that allows the quantification in rules will learn the

following general rule, *If married_to(X, Y) ∧ female(X) Then wife(X, Y)*, where X and Y are variables that can be bound to any person, such as binding X as mary and Y as john in the above case. A knowledge base that includes rules, facts and a type-hierarchy can be generated and interfaced with an inference engine that allows user-interaction and enables greater explanatory capability [Nayak *et al.*, 1997].

2.2.2 Symbolic propositional inductive learning

As described earlier, the methods of rule extraction from ANNs first require training of an ANN on a subset of data for the given problem domain and then generation of equivalent rules from the trained network. Whereas symbolic inductive learning techniques infer the classification rules directly from the presented data, or by learning a decision tree and then translating the tree into an equivalent set of rules. In the attribute-based representation language of rules, each example is described by a fixed set of attributes, where each attribute takes a fixed range of values either discrete (boolean or multi-valued) or continuous. The goal concept is specified as the output of the desired function either a boolean decision or a multi-valued decision. Each generated rule is a series of conditions consisting of attribute-value pairs, followed by a single conclusion that contains the target class and the corresponding value. Mitchell [1997] considered several key dimensions in the design space of such rule learning algorithms. One of the dimensions is *sequential covering vs simultaneous covering.*

Learning decision trees

Decision tree learning algorithms are categorized as *simultaneous covering algorithms* [Mitchell, 1997]. These algorithms learn the entire set of disjunctions simultaneously as part of a single 'general to specific search'. Methods for learning decision trees from examples have been quite popular in machine learning due to their simplicity, efficiency and capability of dealing with a large number of training examples. One of the earliest work on decision tree

learning is Friedman and Breiman's CART system [1977]. In the early 80s, the best known decision tree classifier was Quinlan's ID3 algorithm [1979], which constructs the decision tree by selecting the most informative attribute according to a gain criterion [Quinlan, 1979]. ID3 has been refined to handle continuous attributes, to choose an appropriate attribute selection measure, and particularly to tolerate noise (incorrect decisions) and missing attribute values, resulting in the current versions, ASSISTANT [Cestnik *et al.*, 1987], C4.5 [Quinlan, 1993a] and C5 [Quinlan, 1998].

The decision tree learning algorithms start by constructing a decision tree from top to bottom. Attributes are evaluated at each step to form descendant nodes. The attribute selection is based on a 'statistical test' to determine how well it classifies the training examples. ID3 uses the *information gain* criterion that measures how well a given attribute separates the training examples according to their target classification. This measurement, however, favors attributes with many values over those with few values. C4.5 and C5, the successors of ID3, handle this bias by adopting the new measure called *gain ratio*. The gain ratio discourages selection of attributes with many uniformly distributed values by penalizing such attributes.

While decision tree learning methods are effective in practice, there are a number of limitations. Decision tree approaches do not always produce the most general production rules [Imam and Michalski, 1993]. These methods find rules by fitting shapes of the decision boundaries between the classes formed by lines that are parallel to the axes, yielding rectangular shaped regions [Breiman *et al.*, 1984]. For example, the simple linear relationship of features $X = Y$ (representing a 45 degree diagonal line passing through the origin for separating two classes) is not readily represented by fitting rectangular boundaries. A decision tree method will approximate this behavior by dividing the solution space into successively smaller rectangles, in the form of staircase function. Also, when a decision tree has many layers of nodes, the amount of data that

passes through the lower leaves and nodes is so small that accurate learning is difficult.

Learning concept definitions from examples

There are many algorithms which do not employ decision trees, for instance the AQ family of algorithms [Michalski, 1980], that fall in the category of *sequential covering algorithms* [Mitchell, 1997]. These algorithms utilize disjunction of attribute values to conduct a general-to-specific beam search for each rule, *covering* the positive examples sequentially and rejecting negative examples [Michalski, 1983]. The AQ family of algorithms handles multiple concept learning by considering the training examples belonging to other classes as negative examples and applying the *set covering* algorithm. The sequential algorithm CN2 [Clark and Niblett, 1989] is one of the AQ family that removes AQ's dependence on specific examples during search. CN2 learns one rule at a time, removing the covered positive examples and repeating the process over the remaining training examples. A sequential algorithm like CN2 may be preferred if data is plentiful enough to support the larger number of independent decisions required by the sequential covering.

2.2.3 First-order inductive learning

The two previous ways of learning a set of rules (using rule extraction from ANNs or symbolic propositional learning methods) represent the rules in an attribute-value language. Despite the relative dominance of the propositional formalism, there have always been researchers unsatisfied with its expressive power [Wrobel, 1996]. As a result, a new research area called inductive logic programming (ILP) [Muggleton, 1991; Lavrac and Dzeroski, 1994; Muggleton and DeRaedt, 1994] was evolved from previous research in machine learning, logic programming and inductive program synthesis. ILP lies between logic programming and learning from examples, and aims at inductive learning of logic programs from examples. Like relational ML, it deals with the induction of concepts represented in first-order logical form. This section

briefly summarizes the important facets of ILP. As the target language of the symbolic rules generated by the GYAN methodology is a restricted first-order formalism, and an ILP system FOIL forms a basis for comparison.

The principal differences between zeroth-order (propositional) and first-order supervised learning systems are the form of training data and the way that the learned theory is expressed. Data for propositional learning algorithms comprise preclassified cases, each described by its values for a fixed collection of attributes. These algorithms develop theories, in the form of decision trees or production rules, that relate an example's decision to its attribute values. In contrast, inputs to first-order learners usually contain ground assertions of a number of multi-argument predicates. The learned theory consists of a logic program containing quantifiers and variables, restricted to Horn clauses or something similar, that predicts when a vector of arguments satisfies a designated predicate.

An advantage of ILP over propositional learning systems (ID3, CART, C4.5, AQ) is its capability of utilizing background knowledge. The propositional learning systems do not take significant advantage of existing domain knowledge. Many first-order learning systems have been developed based on ideas that have proved effective in attribute-value learning systems. The LINUS system developed by Lavarc *et al.* [1991] transforms ILP problems to an attribute-value language form, incorporates ASSISTANT [Cestnik *et al.*, 1987] (a member of the decision tree family) and NEWGEM [Mozetic, 1985] (a member of the AQ family) in its learning environment, and finally translates results to Horn clauses. The FOIL system developed by Quinlan [1990] uses the set covering approach as in AQ [Michalski, 1980], a heuristic information-based search taken from ID3 [Quinlan, 1979], and the idea of top-down searching of refinement graphs taken from MIS [Shapiro, 1983]. Michalski's [1983] work on INDUCE included a variant of the system that included first-order elements in a special syntax called the 'Augmented Predicate Calculus'. Though the

concept description language becomes more expressive in ILP systems and the splitting of given positive and negative examples becomes easier, ILP suffers from some shortcomings:

- ILP systems have been restricted to qualitative predictions of activity (high, low etc.) and have yet to deal with numerical values. Handling numbers in ILP has mainly been tackled via transformation of relational problems into propositional ones as in LINUS [Lavrac *et al.*, 1991], or by using adequate 'numerical knowledge', be it built-in as in FOIL [Quinlan, 1990] or provided in declarative form as in PROGOL [Muggleton, 1995]. When continuous attributes are present in the problem domain, ILP systems have not been able to better the performance of standard quantitative analysis techniques like linear regression [Srinivasan and King, 1996]. ILP systems, however, perform creditably when such attributes are not present.

- Induction in first-order logic languages suffers from additional complexity as compared to induction in attribute-value languages because of the large number of possible matchings between a candidate hypothesis and a training example. ILP suffers from an infinitely large search space and a more complex underlying inference mechanism [Furnkranz and Pfahringer, 1998]. Learning of universally quantified concepts may become very complex, as learning of existentially quantified concepts has already been proved to be an NP-complete problem [Haussler, 1988]. The predictive power is worse when switching from propositional to first-order logic learners. ILP systems face a serious degradation of performance when moving from given examples to test data [Bergadano and Gunetti, 1995]. As a result, ILP algorithms sometimes perform worse than the propositional algorithms on the learning tasks where a propositional description suffices [de Mantaras and Armengol, 1998]. ILP systems are usually slower than their propositional counterparts [Lavrac and Dzeroski, 1994].

In the emerging field of ML in the 1970s, several approaches[7] to learning in first-order representation were developed for induction of structural or relational descriptions. One of the earliest methods is Patrick Winston's [1970] work on learning concepts of arches, which employed a relational representation based on graphs. A graph representation was also used by Steven Vere [1975] to learn relational productions, in part building on Plotkin's ideas [1970; 1971]. ILP systems mainly follow two strategies to explore the hypothesis space: bottom-up (specific to general); or top-down (general to specific).

Bottom-up strategy

The bottom-up strategy is based on generalization of a set of positive examples from specific to general. The first formal method developed for inductive generalization is Plotkin's [1970; 1971] framework. Plotkin selects two positive examples of a target concept and attempts to find the least general generalization (lgg). Most of the research in the area of ILP refers to Plotkin's framework in order to define the hypothesis space in which the search for the concept descriptions is performed. Buntine [1988] extends the concept of lgg to relative least general generalization (rlgg) by using background knowledge. He further proposed a method to compute the most specific generalization (msg) that is essentially the one used in GOLEM [Muggleton and Feng, 1990]. Further, Muggleton and Buntine [1988] introduced the notion of inverse resolution in first order logic. They argued that inverse resolution is complete for induction, since the resolution rule is complete for deduction. Based on this hypothesis, they proposed four inverse resolution rules: absorption, identification, intra-construction and inter-construction and implemented these in the CIGOL system [1988]. CIGOL builds clauses from given examples by using these rules and a background theory (which may be empty).

[7] A detailed overview of first-order learning methods of the 1970s period can be found in [Dietterich and Michalski, 1981].

Top-down strategy

Many of the top-down algorithms employ a greedy covering approach to the construction of hypotheses. The learner searches in the space of literals (constructed from training examples and the background knowledge) to fit in a single clause (generally starting with the empty clause) which covers a subset of positive examples, and is 'best' with respect to some quality criteria. Once that specific clause has been found, it is added to the hypothesis, and the positive examples it covers are removed from the set of training examples. This process is repeated until the hypothesis covers all the positive examples.

One of the well known ILP system using the top-down approach is Shapiro's Model Inference System (MIS) [1983]. MIS could not treat complex problems because of its way of searching for hypotheses and failing to overcome the problem of combinatorial explosion. More recently, Quinlan has developed the FOIL system [1990] that learns definite Horn clauses from data expressed as relations. FOIL has proved to be an effective and efficient method on several learning tasks. The FOIL system has been refined by many algorithms such as FFOIL, FOCL [Pazzani and Kibler, 1992] to define and exploit background knowledge in the inductive learning process. While FOIL is a very efficient system that produces good solutions in many cases, due to the heuristic nature of its search there is no guarantee that an existing solution will eventually be found. On the other hand, it has been empirically determined that except for the most trivial problems the space of Horn clauses cannot be searched exhaustively in reasonable time [Wrobel, 1996]. The rules induced by FOIL are overly specific as imposed by the coverage of positive examples.

2.3 Learning a set of rules: Summary and discussion

Based on the discussion of rule-extraction techniques in the earlier part of this chapter, it can be inferred that most of the rule-extraction algorithms

suffer from significant limitations such as lack of generality, scalability, predicative accuracy, rule comprehensibility, etc. A comparative study to find the *optimum rule extraction technique for a given situation* was performed by Andrews *et al.* [1996]. The results obtained from this initial exercise did not reveal a superior rule extraction technique for each category of tasks. The results did, however, point out the need of a technique to represent rules in more expressive language, such as first-order predicate logic or a subset of this.

There has been one more relevant issue with regard to learning a set of rules from data, and that is the question of whether to use connectionist or symbolic techniques. Given that the input to the entire process of rule-learning is a data set and the output is a rule set, it is natural to ask why symbolic induction techniques should not be used instead of training an ANN on the data and then using rule-extraction techniques to extract rules from the trained network. As a result, a number of studies have been conducted to compare the performance of various machine learning algorithms.

In the early 1990s, Project StatLog carried out a comprehensive comparative study of different machine learning, neural and statistical classification techniques [Michie *et al.*, 1994]. About 22 different algorithms were evaluated on more than 20 different datasets of industrial interest. Miller *et al.* [1990], Shavlik *et al.* [1991], Quinlan [1993b] and Andrews *et al.* [1996] undertook a detailed comparison of learning of rule sets by neural techniques and by conventional symbolic induction techniques. It is interesting to note that no particular algorithm could be considered 'best' when considering all performance measurements. The results indicate that one particular algorithm could provide superior predictive accuracy than others with some problem domains, but not for all problems. The results obtained from the studies were regarded as being inconclusive, except for the finding that 'the performance of various symbolic and connectionist algorithms is domain dependent'.

Thornton [1992] further shows with examples that a training set not exhibiting statistical regularities (that is, probability relationship between particular configurations of input values and particular target output values is high) is very hard for any inductive learner (such as ID3, backpropagation, quickprop and cascade ANNs) to deal with. According to Quinlan [1993b], connectionist techniques work better for parallel tasks (where all the input variables are relevant to the classification) and symbolic techniques are more suitable for sequential tasks (where the relevance of a particular input variable depends on the values of other input variables). In general, the predictive accuracy of machine learners is highly dependent on the way in which the training set is processed (sequential or simultaneous) by the heuristics. The selection of a learning algorithm is always relative to a given task, as each technique has its own competence and efficiency such that one may work when others fail.

On the issue of knowledge representation, there is a tradeoff between comprehensibility (good expressiveness of language) and performance (predictive accuracy, computational complexity, etc.) when deciding whether to use first order inductive learners or propositional learners [Lavrac and Dzeroski, 1994; Bergadano and Gunetti, 1995; de Mantaras and Armengol, 1998]. Many ILP systems constrain the language in order to reduce computational complexity, for example background knowledge and hypotheses may be required to be function free (*i.e.* only using variables and constants as terms). In the field of Database, such languages are called Datalog languages (function free Horn clauses) [Wrobel, 1996]. The primary advantage of Datalog languages is that implication is decidable [Wrobel, 1996]. Function free languages are found to be adequate in many practical problems [Quinlan and Cameron-Jones, 1995; Rouveirol and Puget, 1989], and are a good compromise between attribute-value and first-order logic languages.

In summary, although there is a wide variety of algorithms available for learning rule sets from data, there appears to be *no best way to learn rules.*

The algorithms suffer from limitations in terms of rule comprehensibility, performance, generality etc. The purely numerical connectionist networks are inherently *deficient in abilities to reason well*; the purely symbolic logical systems are inherently deficient in abilities to represent the uncertain, approximate, and analogical links -important *'heuristic connections'* between concepts - that are needed to make a new hypothesis [Minsky, 1990]. Versatility can be found in large scale architectures that can exploit and manage the advantages of several types of representations, and each can be used to overcome the deficiencies of the others. In order to get around the limitations associated with rule-learning algorithms, hybrid systems should be developed that (1) combine the expressiveness and procedural versatility of symbolic representations with the fuzziness and adaptiveness of connectionist systems, and (2) utilize the well-established ideas of propositional techniques to represent rules in a constrained first-order language. The following chapter presents a methodology that addresses these limitations by combining many algorithms in its environment and by presenting results in an expressive subset language of first-order logic.

2.4 Chapter summary

This chapter provides the context to understand aspects of the GYAN methodology that appears in the following chapters. The preceding discussion on inductive generalization has laid the foundation for mapping propositional to restricted first order logic rules in the GYAN methodology. This chapter has also reviewed the several ways of learning rule sets. In particular rule-extraction techniques from ANNs are described in detail and it is shown the need to represent rules in more expressive languages than propositional logic.

Chapter 3

Neural learning and rule processing

The previous chapter focuses on learning of a set of rules from the given examples. This chapter extends the idea of learning rules to an integration of neural learning and rule processing such that the integration provides user interface and explanation capability.

The first section of this chapter briefly summarizes a few (representative) connectionist semantic networks since one such system, SHRUTI, is considered as an end user system for the GYAN methodology introduced in the following chapters. The second section of this chapter introduces the idea of the integration of neural learning and rule processing in detail. This framework may be viewed as a special case (or an example) of the GYAN methodology. The symbolic rule processing in this framework is a query based learning approach in which the identified attributes and outputs (predicted by the trained ANN) are turned into rules with the use of a symbolic learner.

3.1 Connectionist semantic networks

This section summarizes a few connectionist semantic networks. The ability of the human brain to represent and reason in a systematic, rapid and effi-

cient manner has always attracted researchers. Many researchers have tried to mimic the computational processes of the brain by realizing artificial neural networks (ANNs) and connectionist semantic networks (CSNs). CSNs (or localist networks) and ANNs (or distributed networks) are composed of many interconnected weak processing elements that work simultaneously to achieve the outcome. In comparison with ANNs, CSNs are sparsely connected and the network structure is prespecified to suit a particular task [Feldman *et al.*, 1988]. Generally, each individual unit of a CSN (or localist) network can be semantically interpreted.

Fahlman [1979] proposed the design of a marker-passing machine (NETL) consisting of a parallel network of simple processors and a serial computer that controls the operation of the parallel network. Fahlman solved a class of inheritance and recognition problems efficiently by computing transitive closure and setting interconnections in parallel. Some improvements were done on the NETL design, but they suffered from serious semantic problems such as race conditions and cross-talk. The neurally implausible NETL system was never implemented in hardware.

Feldman and Ballard [1982] introduced the first general connectionist model and considered how it might be used in cognitive science. They developed a model of individual nodes, and introduced a method of passing symbolic information from node to node without cross-talk in a biologically plausible manner. Feldman [1982] described the problem of dynamically associating any element of a group of N entities with any element of another group of N entities using an interconnection network. He addressed the variable binding problem to solve this association task with an interconnection network having only $4N^{3/2}$ nodes. Feldman [1986] argued that if the human brain can reason efficiently, then its connectionist formulation should yield a better characterization of higher level thought than symbolic logic. He specifies the internal structure of relations defining concepts by hierarchically compact representations; that is,

relations among concepts are represented by links connecting the representations of the concepts involved.

Shastri [1988] built upon Feldman's ideas about representation into a reasoning system for inferencing and developed a connectionist semantic network to solve certain inheritance and classification problems in time proportional to the depth of the conceptual hierarchy. The system finds the most appropriate answers to inheritance and recognition queries by combining the information encoded in the semantic networks. This is done by controlling the flow of activity of the nodes at each step of processing using distributed mechanisms without a central network controller. He made improvements over the basic model relating to the computational effectiveness of reasoning [1990] and the relevance of connectionism to AI [1991]. Shastri [1992] pointed out the need for a massively parallel realization of semantic networks and showed that structured connectionism offers an appropriate computational framework for doing so.

The use of massive parallelism is essential when developing a realistic model of cognitive processing which responds in human-like time scales [Shastri, 1988]. Connectionist inference systems execute reflexive reasoning by exploring a number of inferential paths concurrently. Simultaneous processing of these paths is only possible with parallel mapping. In principle, parallel mapping of semantic networks involves assigning a simple processor to each node in the semantic network and representing each link in the network by a hardwired connection between processors. Current technology does not support direct realization of such large and highly interconnected networks of processors. An alternative approach is to simulate semantic networks on existing general purpose parallel machines. The use of massive parallelism in large scale intelligent systems will reduce run time and make them more feasible for use in realistic applications. There are several levels of parallelism possible in CSNs, while mapping on parallel machines: (1) rule-based parallelism in which multiple

rules are allowed to fire simultaneously, (2) fact-based parallelism in which all the facts associated with a predicate may be matched simultaneously, (3) query-based parallelism in which multiple queries can be posed simultaneously.

Shastri and Ajjanagadde [1993] described a computational model (SHRUTI) addressing some of the cognitive science challenges and resolving the artificial intelligence paradox (see the following chapter for a detailed description). There have been many advancements in the original SHRUTI model resulting in the enhanced model [Shastri, 1999]. SHRUTI serves as the representative connectionist model for the end user system in the GYAN methodology presented in the following chapters. Mani and Shastri [1995] implemented SHRUTI on existing general purpose massively parallel architectures such as Connection Machine CM-2 (a SIMD, Single Instruction Multiple Data machine) and Connection Machine CM-5 (a MIMD Multiple Instruction Multiple Data machine). In SHRUTI-CM5, the partitioning is at the level of knowledge base elements such as concepts, predicates, facts, rules, etc. The system runs asynchronously in that each processor continues with its processing irrespective of the progress made by other processors. It currently supports only backward reasoning, not forward reasoning or the integration of forward and backward reasoning. It also does not support handling of negated knowledge or of inconsistency in reflexive reasoning.

Waltz and Pollack [1985] developed a natural language processing system with modular knowledge representation but strongly interactive processing. The system offered insights into a variety of linguistic phenomena and allowed easy testing of many hypotheses. They described a parallel model of representation of context and of the priming of context using massively parallel parsing.

Touretzky and Hinton [1988] developed the distributed connectionist production system, DCPS, to describe the problem of rule based reasoning within a connectionist framework. Their connectionist model consists of many sim-

ple, richly interconnected neurons like computing units that cooperate to solve problems in parallel. In the reasoning process DCPS selects and applies a single rule at each step, and only deals with rules containing a single variable. It was not an efficient model because of the similarity to traditional serial production systems at the knowledge level.

Lange and Dyer [1989] developed an inference system, ROBIN, that is essentially equivalent to SHRUTI. It uses signatures instead of phases to encode dynamic bindings. The units in ROBIN are standard units in terms of connectionist systems, but signatures are sent along the connections in the network. The use of signatures (or integers) in the signals instead of the temporal sequence (or oscillatory behavior) used in SHRUTI weakens the biological plausibility of the ROBIN system.

Hőlldobler [1990] developed the connectionist inference system CHCL to deal with Horn logic. CHCL can do the inferences that SHRUTI and ROBIN are not able to perform, but it is biologically implausible. CHCL is a connectionist system in the sense that each formula is represented by an ANN and the (un-)satisfiability of the formula is determined by spreading of activation through the networks.

Barnden and Srinivas [1991] proposed a connectionist production system (CONPOSIT) which is an architecture for the modeling of complex reflective reasoning processes based on relative position encoding and pattern-similarity association. Argument bindings are propagated by a connectionist interpreter that reads the contents of registers and updates them. This system is more appropriate for modeling of reflective reasoning rather than for reflexive reasoning [Barnden and Srinivas, 1991].

Pinkas [1991] developed a symmetric ANN that expresses first order logic and relies on the energy minimization of symmetric or stochastic networks to

solve the inferencing problem. Pinkas shows that the problem of finding a global minima of an energy function is equivalent to the satisfiability problem in propositional logic.

Moldovan *et al.* [1992] developed the Semantic Network Array Processor (SNAP) to represent and reason with semantic networks using associative memory and marker passing. SNAP provides a special instruction set for network creation and maintenance, marker creation and propagation, logic operations and search/retrieval. A SNAP prototype has been built and is used to implement memory-based parsers in parallel. But SNAP-based systems use special purpose hardware, can only deal with semantic networks and do not support the full range of inferences such as those supported by SHRUTI.

CONSYDERR [Sun, 1994] is a two-level connectionist system that uses a localist network (CSN) to perform rule-based reasoning and a distributed network (ANN) to encode feature similarities. The two levels interact to provide a robust system for cognitive modeling of commonsense reasoning and decision making. This is a tightly coupled multi-module system in which each node in one module (ANN) is connected to a corresponding node in the other module (CSN). Based on CONSYDERR, Sun and Peterson [1997] presented a more complete and integrated architecture CLARION. It consists of two levels: (1) the bottom or reactive level which contains procedural knowledge acquired through connectionist reinforcement learning; and (2) the top or rule level which contains declarative knowledge acquired through rule extraction. With the integration of ANNs and CSNs, CLARION is able to acquire and represent dynamically generic knowledge level at the top level, and explicitly access and communicate that knowledge.

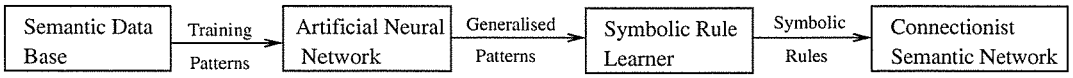


Figure 3.1: The hybrid rule generation process

3.2 A framework for the integration of neural learning and rule processing

ANNs have the ability to learn and generalize the relationship between input-output pairs, given a set of examples consisting of inputs and the corresponding outputs. Knowledge in an ANN is distributed across the architecture (input/hidden/output nodes and interconnections) and encoded in terms of numerical weight parameters. They do not have explicit, symbolic and declarative knowledge structures that allow the representation of explanation structures [Diederich, 1992]. On the other hand, connectionist semantic networks are capable of representing complex and embedded knowledge structures. Knowledge in a CSN is expressed in terms of concepts (or variables and their fillers) and the hierarchical relationship between concepts. Each concept is represented by a node or a group of nodes, and the relationship between them is represented by an is_a link [Shastri, 1988]. CSNs do not have a mechanism to adapt the knowledge base.

The shortcomings of both types of systems can be overcome by building an integrated system (Figure 3.1) that takes advantage of the strengths of both [Nayak, 1996]. The integrated system overcomes the disadvantage of CSNs that they do not have integral learning to build knowledge bases, by automating the knowledge base generated by ANNs. The integrated system overcomes the disadvantage of ANNs that they are not able to represent complicated data structures and explanation, by integrating a CSN as an end user system. The integrated system learns a problem by using an ANN, and makes the learned knowledge available to users by transferring it into a CSN. The important task is to perfectly map the knowledge learned by an ANN into a connectionist rule-based reasoner.

3.2.1 Symbolic rule mapping

The rules from trained ANNs are not recovered by rule extraction but simply by querying the networks. This method of learning propositional rules uses a feedforward ANN along with a symbolic induction algorithm. The basic idea incorporated in this rule-extraction technique is:

- An ANN is trained for a problem domain. If applicable the input space is then reduced by a pruning process and a compressed dataset (reduced in dimension and size) is obtained.
- Once the training is completed, the (pruned) network is queried to gain new (generalized) information. The selection of patterns that are used as queries are important for the formation of rule sets. In principle, the query patterns should contain all the information that the ANN has really learnt. The network's response is recorded along with the input patterns.
- The input attributes and the decision attribute(s) (the response of the network) are then fed to a symbolic induction algorithm to generate the corresponding concept definitions.

The obtained propositional rules can be transferred into a representation of facts, a type-hierarchy and rules with variables and n-ary predicates using a mapping algorithm, and the generated symbolic knowledge base can now be used in a connectionist knowledge representation system or any inference engine.

3.3 An example: From simple to complex rules

The process is demonstrated by a simple example which consists of information about the relationships among categories and their properties [Rumelhart, 1990]. This discrete example is chosen because it is easier to demonstrate the 'logic' of generating simple to complex rules by this example. This example is

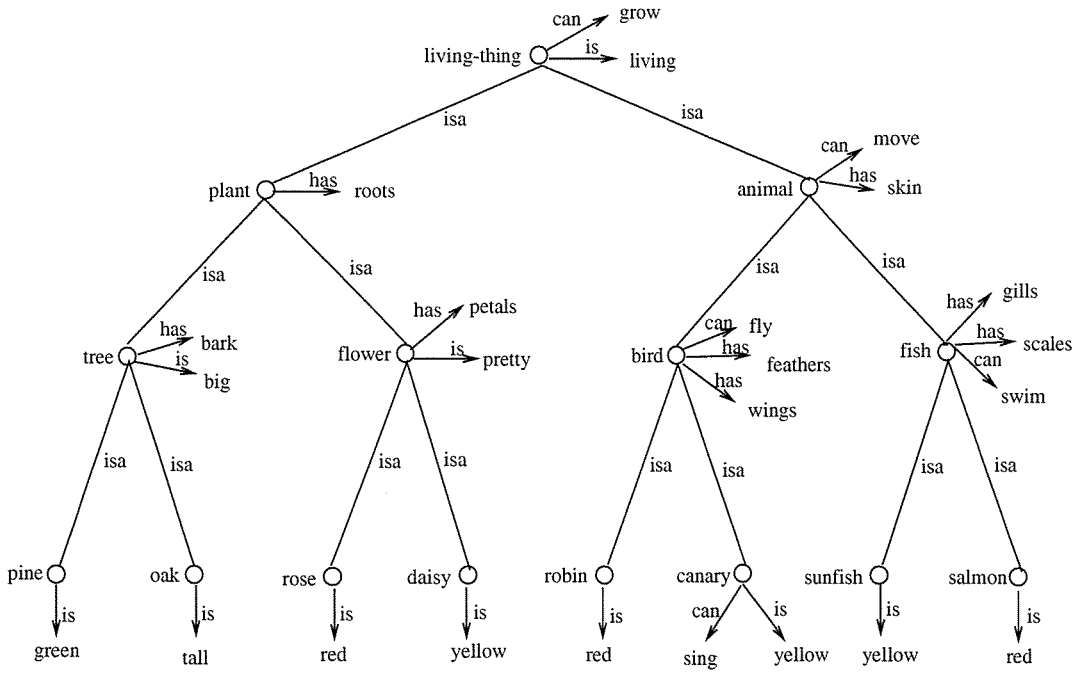


Figure 3.2: A semantic data network showing relationships among various plants and animals [Rumelhart, 1990]

a data structure, a semantic network, that summarizes a number of facts as illustrated in Figure 3.2.

A feedforward network with two hidden layers introduced by Rumelhart [1990] serves as the representative ANN model for learning and generalization of the semantic network. The Rumelhart's PDP network is chosen because of its special architecture (Figure 3.3) that assists to retain the symbolic nature of the problem (semantic network) during learning and is further used in reasoning.

A symbolic learning algorithm developed by Orlowski [1995], OSL, serves as the symbolic learner for learning of propositional rules. The OSL algorithm is selected as it does not incorporate any type of heuristics to generate rules unlike other symbolic inductive learners such as C4.5 [Quinlan, 1986] or AQ15 [Michalski, 1980].

A connectionist knowledge representation and reasoning system, SHRUTI [Shastri and Ajjanagadde, 1993], serves as the representative connectionist model.

SHRUTI is chosen as an end user system (for the GYAN methodology) because of the expressiveness of SHRUTI approximates (restricted) first-order logic representation that is capable of representing complicated data structures and explanation. Pinkas type networks [1991] are an alternative to SHRUTI because of their equivalent expressiveness of language. At present Pinkas networks are too slow to be used for practical purposes and more importantly, intermediate steps in reasoning do not necessarily contribute to an 'explanation structure'. Prolog or a symbolic rule-based system can also be used as an end user system, but one of the goal of this thesis is to provide the capability of adapting (rather than manually creating) the knowledge base into a CSN where no such ability is available.

The Rumelhart's network is trained on the available domain. Then the symbolic learning algorithm is used to generate propositional rule sets by analyzing the responses obtained from querying the trained ANN. These propositional rule sets are mapped into a generalized rule sets (having predicates and variables), facts and a type-hierarchy, and then processed by the SHRUTI connectionist reasoning system. This integrated system (Figure 3.1) can in turn be used for deduction and analytical learning [Diederich *et al.*, 1998].

3.3.1 Rumelhart's PDP network

Rumelhart [1990] introduced a feedforward network architecture with two hidden layers for discovering the internal representation within functionally similar patterns (for example, the representations of *rose* and *daisy* must be more similar to one another than to either *pine* or *sunfish*). A special architecture (Figure 3.3) is needed to retain the symbolic nature of the problem (semantic network) during learning for parallel distributive processing (PDP). The distinguishing feature of PDP models is that the representation is embodied as the pattern of activity and distributed over hidden layers in the model. In most PDP models, it is not possible to attach any clear meaning to the activity of an individual unit in the network. In these cases, each unit represents not

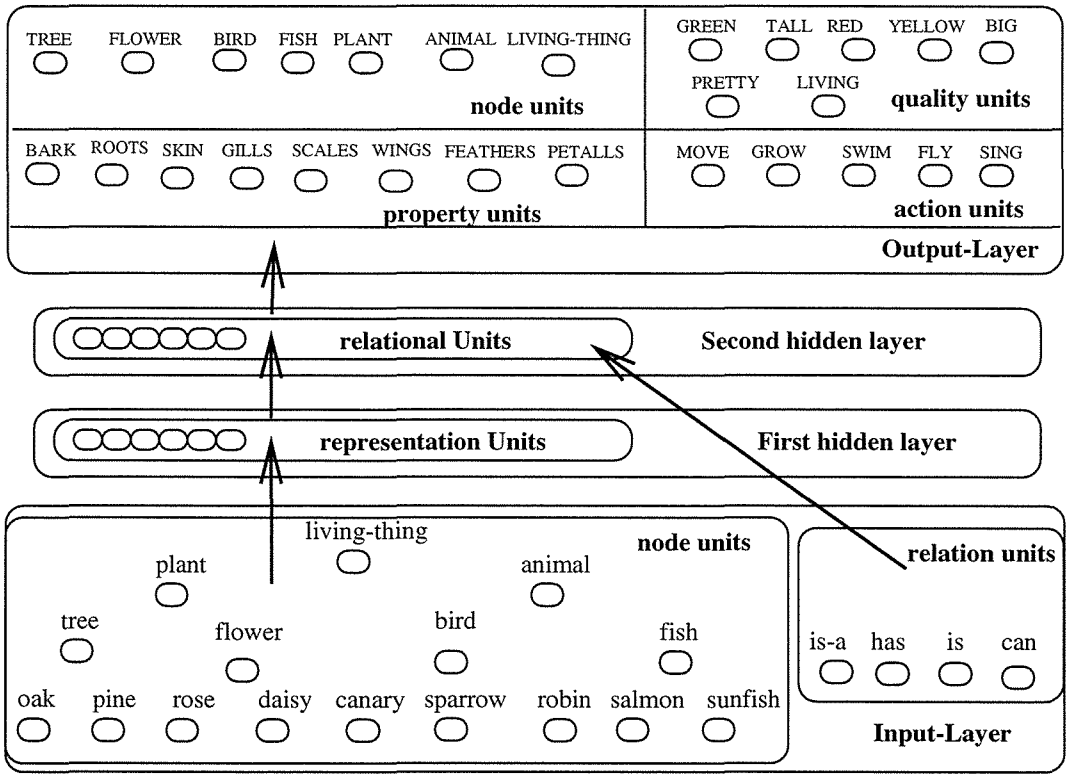


Figure 3.3: Rumelhart's [1990] PDP network, adapted from [Diederich, 1991]

only a familiar concept but also some abstract aspect or relationship, which contributes to the more readily intelligible feature that is represented by the overall activity of patterns.

The architecture

The input layer is partitioned into two sets of units. The first set of units contains units corresponding to concepts (illustrated as ellipses in Figure 3.2) in the semantic network. In the second set, there is one unit for each relation, *i.e.* one unit for each labeled arc of the semantic network. The network knows four relations: 'is-a' (subclass/superclass), 'is' (quality; e.g., something is red), 'has' (property; e.g., a bird has feathers) and 'can' (actions; e.g., a bird can fly). [Diederich, 1991]

The network has two groups of hidden layer units. The first layer labeled as 'representation units' receives inputs only from the 'node units' in the input

layer (*i.e.* the first set of units). The second layer of hidden units ('relational units') receives inputs from the 'representation units' and also from the 'relational units'. The reason for this configuration of hidden units is to develop a representation of the inputs that stimulates similar responses for similar representations, no matter how similar or dissimilar the actual input patterns are [Rumelhart, 1990], and which therefore forms the representation of the distributed concept. [Diederich, 1991]

The output layer of the connectionist network is divided into four groups corresponding to each arc type. The first set, 'node units', consists of sub/super concept nodes (something *is-a* <super-concept>). The second set, 'quality units', consists of units for each quality (a concept node *is* something). The third set, 'property units', consists of units for each property (a concept node *can* have something). The fourth set, 'action units', consists of units for each action (a concept node *can* do something). All the output units receive inputs from the 'relational hidden units'. [Diederich, 1991]

Training

The network is trained by randomly presenting facts chosen from the semantic network data base using the backpropagation learning algorithm [Rumelhart *et al.*, 1986]. The initial weight limit is set to 0.25 with random initialization of weights. The various learning rates of 0.0001, 0.01, 0.1, 0.15, 0.2, 0.4 with a fixed momentum of 0.9 are used to train a number of networks. Figure 3.4 shows the best results which were obtained using the learning rate of 0.1, the sum-squared error on training set is reduced to 0.00826. For example, the units for 'canary' and 'can' are clamped on in the input layer and the network is trained to turn on the units for 'sing', 'fly', 'move' etc. in the output layer. The network has been able to store all the desired information by creating internal representations as conceptual structures after the presentation of all the patterns.

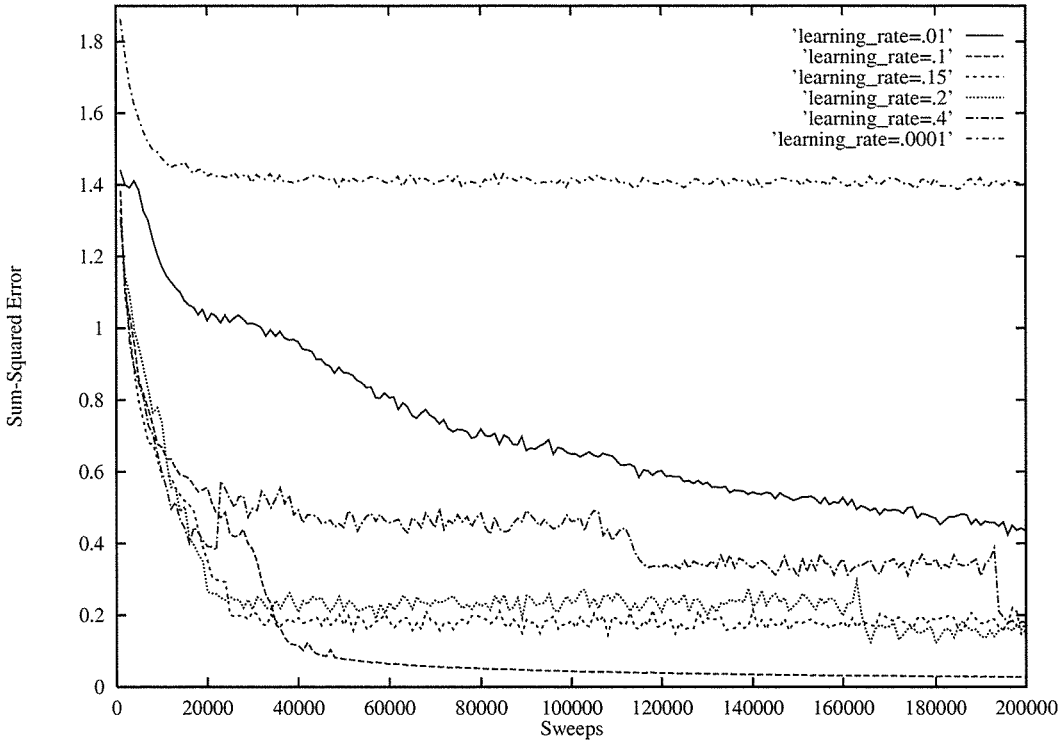


Figure 3.4: Learning curves during training, momentum=0.9

Generalization

Rumelhart [1990] analyzed two aspects of the network: (1) the representation over the 'representation units', and (2) generalization. The generalization task is described in more detail as it is more relevant to the purpose of the framework. In order to generate queries, the network is presented with new patterns that it has not seen before. For example, the network is trained for patterns such as 'sparrow isa bird' and 'bird can move, grow, and fly'. Now to know, *what can bird do?*, the network is asked by turning on the input units 'sparrow' and 'can'. The network responded with 'fly', 'move' and 'grow'. The sum-square error during generalization is 0.16. For the purpose of rule generalization, the network has been queried to gain all the information about the type, quality, action and properties between concepts.

The advantage of this type of network is that new information can easily be added to the trained network and many new inferences readily made [Rumelhart, 1990; Diederich, 1991]. For example, *'lotus isa flower'* is a new fact that

the network has not seen before. The input nodes 'lotus' (a new node) and 'isa' and the output node 'flower' will be turned on to learn this new information. During this training period, all the other 'node units' connections will be frozen except those from the 'lotus' unit to the representational units . After learning, when the network is asked 'lotus has', the network responds with 'petals'. The newly learned information reflects all the properties that a flower has.

3.3.2 Symbolic inductive learner

Essentially, any symbolic inductive learner can be used to derive symbolic rules from the set of inputs and decision attributes generated by querying the ANN. However, the algorithm proposed by Orlowski [1992] is used to determine object identification rules. The reason of choosing this algorithm for this problem domain is that it does not incorporate any type of heuristics to generate rules and its inherent ability to absorb new knowledge.

The algorithm is based on a dynamic approach that employs a method of maintenance of a rule base by the (minimal) modification of 'existing rules' whenever a new instance is presented. The algorithm takes a set of attribute values for each pattern and the corresponding decision attribute(s) computed by the trained ANN. The algorithm (1) updates the rules for classifying each existing pattern examined to date in the presence of a new case if the algorithm does not distinguish between the existing patterns and the new case, (2) determines the rules by classifying the new case in the presence of the existing patterns if the new case is distinct from other existing patterns, and (3) removes all rules that are not minimal.

Rule generation

The algorithm described by Orlowski yields a rule set when presented with a set of attribute values for each pattern generated, together with the decision computed by the trained ANN for each pattern. The network's response along

with the input values is recorded for each query. The generated patterns are then rearranged to convert this data structure (a semantic network) into a decision problem. The nodes in the Rumelhart network (Figure 3.3) corresponding to the 'relation units', 'quality units', 'property units' and 'action units' become the newly formed input attributes in the decision problem. The decision (dependent) attributes are the corresponding 'node units'. The input 'node units' are classified into four decision classes: class0 : 'tree' (oak, pine); class1 : 'flower' (rose, daisy); class2: 'bird' (canary, sparrow, robin); and class3 : 'fish' (salmon, sunfish). A total of twenty-nine (29) such cases were presented to the symbolic learner to identify the relationships between objects. For the given data set, a total of fifty-nine (59) rules were generated of which 22.03% define the extent of class bird, 16.95% relate to class fish, 49.15% to class flower and the remaining rules to class tree.

3.3.3 SHRUTI knowledge base

Construction of a SHRUTI knowledge base, consisting of is_a relationships, facts and rules fulfilling the imposed restrictions (chapter 4, section 4.4.2), has been generated for the purpose of querying and explanation. Some of the is_a relationships in the generated knowledge base for the classification of various plants and animals, are:

is_a(animal,living-thing)	is_a(bird,animal)	is_a(canary,bird)
is_a(plant,living-thing)	is_a(tree,plant)	is_a(pine,tree)

Some of the rules, now quantified by prior knowledge, are:

$\forall$ X [bird(X) $\Leftarrow$ $\neg$has_roots(X) $\wedge\neg$has_scales(X) $\wedge$ can_grow(X)]

$\forall$ X [flower(X) $\Leftarrow$ $\neg$can_fly(X) $\wedge\neg$can_swim(X) $\wedge$ can_grow(X)]

$\forall$ X [fish(X) $\Leftarrow$ has_skin(X) $\wedge$ can_swim(X) $\wedge\neg$has_wings(X)]

$\forall$ X [tree(X) $\Leftarrow$ is_living(X) $\wedge$ has_roots(X) $\wedge\neg$has_petals(X)]

The knowledge base also contains facts and queries. Facts are the instantiated antecedent predicates of the rules described above. The queries are

the instantiated consequent predicates of the rules. A query can have either constants or variables or existential variables (denoted by a wild card !) or combinational. Once the knowledge base is created, the SHRUTI network is generated by using the SHRUTI simulator [Hayward, 1999] and several queries are posed. The queries are answered with 'yes' or 'no' within a few milliseconds via backward reasoning. Some of the posed queries are:

Query	Answers (yes-no form)	Facts
? bird(robin)	Yes	can_fly(robin)
? flower(living-thing!)	Yes	can_grow(daisy)
? tree(oak)	Yes	has_bark(oak)
? fish(sunfish)	Yes	can_swim(sunfish)
? fish(canary)	No knowledge of relation ship being true or false	-

3.3.4 Discussion: The example

In this example, the ANNs' knowledge is acquired by training and then querying the network. As in any query based approach, the quality of the generated rules very much depends on the contents of the queries. The queries should be such that the abstract relationship between concepts learned by the ANN is captured completely. A number of query based rule extraction technology have been developed to efficiently extract rules from ANNs. For this particular problem domain, the human knowledge of domain plays an important role in selecting queries to form rules, and the symbolic learner is capable of recovering these refined concepts in a symbolic form. The generated knowledge base can then be interfaced with a connectionist knowledge based reasoner to provide a symbolic explanation. A special characteristic of this example framework is incremental maintenance of an existing knowledge base in the presence of new information due to incorporation of the special type of network architecture and the symbolic inductive learner.

3.4 Discussion: Integration of neural learning and rule processing

Recent advances in artificial intelligent systems using ANNs and knowledge based systems show the potential cross-fertilization between these two approaches based on the ways of addressing the issue of knowledge representation. The neural approaches differ from the knowledge based approaches in several respects. The ANN learning approaches assume that intelligence emerges through interactions among a large number of highly interconnected but relatively simple processing elements. On the other hand, knowledge based approaches assume that intelligent behavior is derived from knowledge structures. The knowledge of an ANN is embedded in connections among nodes and associated weights. The knowledge of an expert system is often represented in rules and implemented through knowledge engineering or machine learning techniques. Learning in knowledge-base systems generally relies on the quality of initial knowledge.

An ANN processes information by propagating and combining activation through the nodes, but a knowledge based system reasons through symbol generation and then pattern matching. The knowledge based approach emphasizes knowledge representation, reasoning strategies and the ability to explain, whereas the ANN approach emphasizes learning from examples. Since the two approaches address different levels of knowledge representation and problem solving, combining their strengths in an integrated environment is justified. One way of combining them is to transform the rule base into an ANN. A number of researchers [Fu, 1989; Towell and Shavlik, 1993; Andrews and Geva, 1996; Hilario, 1997] have presented approaches that generalize the ANN so that it can effectively deal with rule bases which are multi-layered and involve logical conjunctions. This type of approach utilizes ANNs to maintain and refine the knowledge base (which may be noisy, incomplete and incorrect) by recognizing and deleting the incorrect rules. The other way of combining

the two approaches is to use neural networks to learn relationship among concepts and generate categories/rules with a symbolic processing, and use this generated knowledge base to conduct high-level inferencing.

A few domain-specific systems have been developed by adopting the second approach. The SYMCOM system [Wu *et al.*, 1997] integrates a distributed recurrent network (ANN), a localist network (CSN) and a knowledge based symbolic sub-systems to choose a correct meaning for a word with multiple senses. This is not very clear in the paper how the generic (knowledge) rules are generated from the ANN, except that the SYMCOM's knowledge base is hand-built at the present. Ulug [1989] presents a hybrid system in which a neural network component classifies measurements of hydraulic pressure over time, and then an expert system makes diagnostic statements based on the classification. But the method that combines the neural network with the expert system is inefficient. The *SCRuFFY* architecture [Lin and Hendler, 1995] uses a temporal pattern matcher to map neural network's encoded knowledge into a symbolic representation which a rule-based expert systems can reason on. The temporal pattern matcher looks for patterns in the consecutive outputs of the neural network, instead of an output at a single instant in time. A particular distinct output is passed as a fact to the expert system as they are discovered. In general, these systems lack a method that generates the generic and conceptual knowledge embedded in neural networks that can be effectively encoded in exert system.

The GYAN methodology attempts to make a general-purpose integrated system by combining an ANN and a CSN. It transforms the knowledge embedded in the ANN into a form suitable for input to the SHRUTI knowledge base reasoner. Without ANNs (learning from experience, generalization and extraction), the methodology will not be able to acquire generic and conceptual knowledge for modeling and reasoning in CSNs (such as SHRUTI), and therefore a CSN has to rely on a pre-wired and/or externally given knowledge.

Without CSNs (the reasoning and explanation capability), the methodology will not be able to represent generic and conceptual knowledge, and explicitly provide user interface.

3.5 Chapter summary

This chapter introduces the basic idea of the GYAN methodology, *i.e.* the integration of neural learning and rule processing for the generation and representation of generic and conceptual knowledge. With a simple example framework this chapter demonstrates how a rule processing method integrates a neural learning system and a knowledge base reasoner. Essentially, this chapter emphasizes an efficient mapping between the low level symbol mapping (or distributed connectionist networks) and high level reasoning (or localist connectionist networks).

Chapter 4

Gyan: A methodology to generate predicate rules

The discussion in chapter 2 has established the merit of the idea of using first-order predicate logic (with some assumptions) as a language to represent the knowledge embedded in neural networks. That chapter also laid the foundation of a hybrid methodology to inductively generalize rules from data sets, a methodology that combines the adaptiveness of connectionist systems with the expressiveness of symbolic representations by interfacing several learning algorithms. Chapter 3 emphasizes the integration between ANNs and CSNs, and shows how the user interface can be achieved with neural learning and rule processing. The focus of this chapter is to present a domain-independent multi-stage methodology, GYAN, that allows a systematic transformation of data sets into quantified rules with predicates, and subsequently allows user interaction by interfacing with a knowledge based reasoner.

The chapter starts with an overall presentation of the GYAN methodology, giving a more detailed discussion of each step in the following sections. The second section shows how a relationship between attributes is identified when examples, each consisting of a fixed number of attributes, are presented to ANNs utilizing the cascade correlation, BpTower and constrained backpropagation algorithms of neural learning. The third section introduces a prun-

ing algorithm to remove superfluous nodes and links from the trained ANNs. The fourth section shows how the rule extraction methods *LAP*, *RuleVI* and *RULEX* are applied to ANNs to validate the output decisions, and a quantified rule set consisting of predicates is generated that allows for an explanation component in the connectionist architectures. The last section provides a discussion of the user interfacing phase of GYAN.

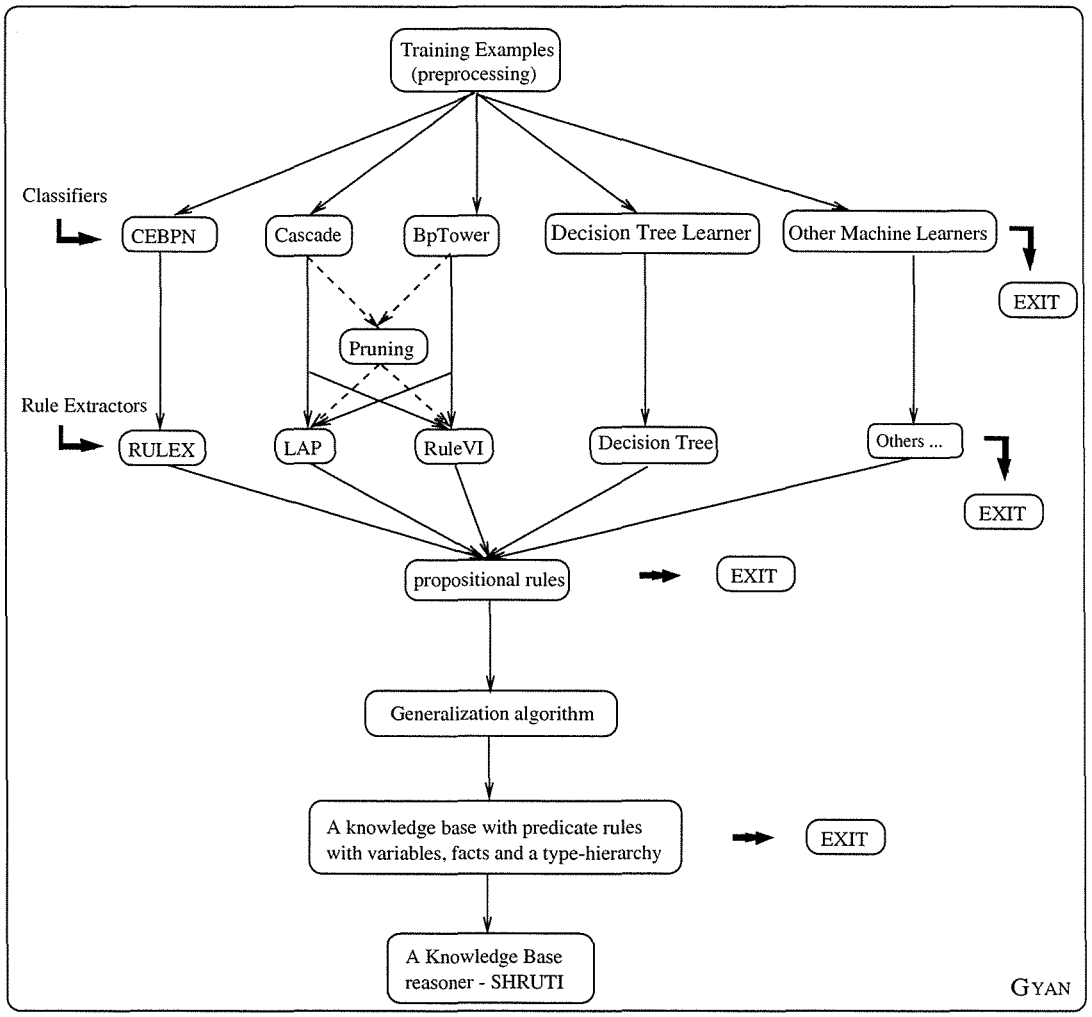


Figure 4.1: An overview of the Gyan methodology. All the incorporated programs used in each phase are shown. The main objective of Gyan is to understand and extend the capabilities of ANNs. Though Gyan can use any propositional rule learner to inductively learn concepts from examples, and enhance the expressiveness of generated propositional rules to (restricted) first-order rules. According to the required hypothesis language (*i.e.* neural network, attribute-value, a subset language of first-order or a knowledge-base) one can exit after any phase.

4.1 The Gyan methodology

The overall objective of GYAN is to form a symbolic generic representation, consisting of predicate rules with variables, from the ANN trained for the given problem. The general framework of GYAN can be stated as follows: given a set of *positive training examples* E^+, a set of *negative examples* E^- and a hypothesis in the form of the *trained neural network ANN*, find the *set of rules consisting of n-ary predicates and quantified variables KR* such that:

$$\forall e_i^+ \in E^+,\ ANN \cup KR \models e_i^+, \text{ and } \forall e_i^- \in E^-\ ANN \cup KR \not\models e_i^-$$

The GYAN methodology starts with the construction and training of feed-forward ANNs for the inductive acquisition of concepts from examples. In the next stage, a clustering based pruning algorithm is applied to remove redundant nodes and irrelevant links from the trained ANNs. The third stage extracts predicate rules from the pruned networks. In this stage, first an equivalent disjunctive normal form (DNF - a disjunction of conjunctions) expression or disjunction of conjunctive normal form (CNF - a conjunction of disjunctions) expressions is gained from the pruned network. Later, the expression is generalized and mapped into predicate rules, which form a knowledge base suitable to the SHRUTI [Shastri and Ajjanagadde, 1993] representation formalism. In the last phase, the generated knowledge bases are interfaced with the connectionist rule-based reasoner SHRUTI for forward and backward reasoning, which enables greater explanatory capability by allowing user interaction.

An illustration of the methodology, including all the incorporated programs, is given in Figure 4.1. The knowledge representation system developed in this thesis offers a choice of more than one language to the user for the expression of domain knowledge for a given problem, such as: (1) the trained ANNs; (2) propositional rule sets; (3) predicate rule sets; and (4) knowledge bases consisting of facts, rules and a type-hierarchy. The steps to learn and generate rules with variables and n-ary predicates from an ANN are summarized in Table 4.1, and discussed in the following sections.

Phase 1. Select and train a network on data from a given problem domain until it reaches the minimum training and validation error.
Phase 2. Prune the network to remove redundant links and nodes.
Phase 3. Generate the representation consisting of a type-hierarchy, facts and predicate rules:
3.1 Extract an equivalent DNF expression or disjunction of CNF expressions from the network.
3.2 Generalize the expressions into predicate rules with variables.
Phase 4. Interface the generated knowledge base with a rule based reasoner to provide a user interface.

Table 4.1: The Gyan methodology

4.2 Phase1: ANN training

Learning is often cast as the problem of identifying and generalizing a hypothesis that describes some already known concepts [Shavlik and Dietterich, 1990]. ANNs using supervised learning algorithms fall into this class of learning systems. ANN learning can be considered as the process of finding a hypothesis that approximates a function that is contained in examples used for training the network. In supervised neural learning algorithms, a training set of pre-classified examples is presented to the network. Each training example is described using a fixed number of attributes for which values are given along with the correct output value(s). The goal is to form a description that can be used to classify previously unseen examples with high accuracy.

In ANNs, the complexity of the hypothesis space is constrained by the user, who initially selects an appropriate network for the given problem. The necessity of *a priori* specification of a network structure makes ANN techniques suffer from serious difficulties [Ishikawa, 1996] such as networks that are too small cannot represent the required function, while networks that are too large are prone to over-fitting which results in poor generalization. Neural learning algorithms that explicitly try to build an ANN by the use of *constructive techniques* or *subtractive techniques* or *regularisation techniques* are an alternative to the problem and help to ease the burden of having no initially specified appropriate network. In constructive techniques, training is started with a

small size network and keeps adding units until an acceptable network state is achieved. In subtractive techniques, training is started with a large size network and eliminates superfluous links and nodes until the network is just able to represent the required function. In regularisation techniques, training starts with a large number of parameters but has a limitation in the size of the parameter dimension by imposing additional constraints such as a weight decay term in error calculation, etc.

The present approach is in general independent of the underlying feedforward network architecture, however, the constructive techniques such as cascade correlation, BpTower and constrained error back propagation algorithms are utilized to dynamically build neural networks for the given problems. A regularisation term, weight decay, is employed during the cascade correlation and BpTower learning of ANNs. When the ANN learning process has been completed, a pruning algorithm is applied to remove redundant nodes and links from the networks trained with cascade and BpTower algorithms.

These ANNs are chosen as they demonstrate a number of architectural and algorithmic differences such as utilizing a bottom-up, or a top-down, or a local function approach to building the network. Use of these ANNs show the feasibility of applying the GYAN methodology to a broad spectrum of ANN architectures and algorithms.

4.2.1 The cascade correlation algorithm

Cascade correlation neural networks are particularly included in GYAN as they have been successfully applied to a diverse range of problem domains [Waugh and Adams, 1993; Collier and Waugh, 1994]. The cascade correlation algorithm [Fahlman and Lebiere, 1990], one of the methods for incrementally building a feedforward network, starts with an input and an output layer with no hidden units. The algorithm constructs the network by initially training the output unit(s) to approximate the target function using gradient de-

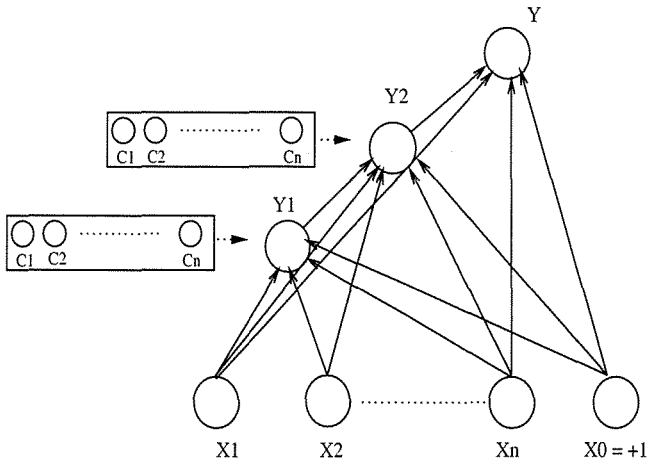


Figure 4.2: A Cascade Correlation network. For simplicity, only one unit in the output layer is shown. Networks with multiple output nodes can be generated by this algorithm.

scent [Rumelhart *et al.*, 1986] or quick prop [Fahlman, 1988]. When training stagnates, a pool of candidate units is trained with connections from all inputs and the previously inserted candidate units (if any) to predict the network error. Each of the candidate units is initialized differently and trained independently of the others. During training, each candidate unit attempts to maximize the magnitude of the correlation coefficient between its output and the network's residual error. When training of candidate units stagnates, the best candidate unit is selected and inserted into the network permanently. A connection is added from the inserted unit to the output unit(s). The weights coming into the newly added unit from the input units (and the existing previously inserted hidden units) are then fixed, and the rest of the network is trained with the newly added unit. This process continues until an acceptable overall network is achieved or the termination criteria are satisfied [Figure 4.2]. The termination criteria are: the root mean square error (RMSE) on the validation (test) set reaches the minimum required error; or the RMS error on the validation (test) set starts to increase; or the maximum permissible number of hidden units has been added.

Weight decay

As training proceeds, some weights begin to grow in order to reduce the error over the training examples. Consequently, the complexity of the learned decision surface increases. The complex decision surface has a tendency to easily fit the existing noise in the training data, or to learn the unwanted or specific characteristics which exist in the training examples (overfitting). A simple method to overcome these problems is to bias the network learning against complex decision surfaces by keeping weight values small. To encourage smooth mappings, a regularisation term is included in the error function to be minimized during training. Adding a *penalty term to the error function* during training is equivalent to associating a cost to each connection in the network [Weigend *et al.*, 1990]. The simplest way of weight decaying is to add a w_{ij}^2 term in the error function. The disadvantage of adding this term is that large weights decay at the same rate as small weights. The goal is to influence smaller weights during training without much effecting larger weights, such that the minimal number of weights are generated and the weights are easily separated into groups. The modified cost function [Weigend *et al.*, 1990] is the sum of two terms based on the rectangular hyperbolic function:

$$\theta(S, w) = \sum_{p \in S} (expected\_ouput_p - actual\_output_p)^2 + \lambda * \sum_{i,j} \frac{w_{ij}^2}{1 + w_{ij}^2}$$

The first term is the standard sum squared error over a set of examples S. The second term describes the cost associated with each weight in the network and is a complexity term. The cost is small when w_{ij} is close to zero and approaches unity (times λ) as the weight grows [Figure 4.3]. Initially λ is set to zero and gradually increased by small steps until learning improves. The learning rule then updates weights according to the gradient of the modified cost function with respect to the weights. The updated weights [Weigend *et al.*, 1990] are:

$$w_{ij} = w_{ij} + \Delta w_{ij} - decay\_term$$

$$decay\_term = \lambda * \frac{2w_{ij}}{(1 + w_{ij}^2)^2}$$

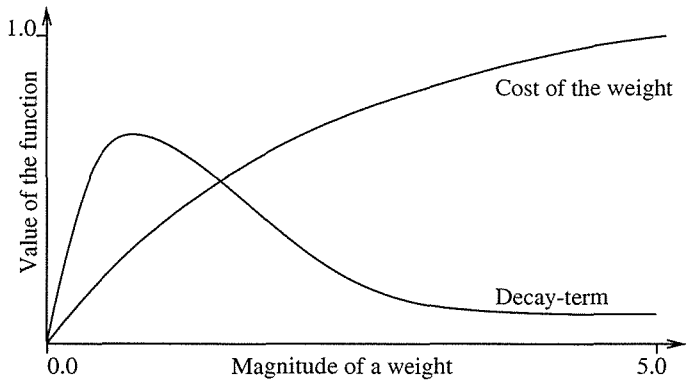


Figure 4.3: Cost for a weight and its derivative with respect to weights in the network [Weigend *et al.*, 1990]. The cost is small when the magnitude of a weight is close to zero, and approaches unity as the weight grows. The value of the decay-term is large when the magnitude of the weight is small, and the value of the decay-term decreases as the magnitude of the weight increases.

The decay term allows smaller weights to decay faster as compared to larger weights (Figure 4.3). Also all the weights except bias weights are constrained at two digits after the decimal point. The fundamental consideration behind these constraints is to generate the networks with weight values that are clustered into a number of small groups, instead of generating uniformally distributed weights. These constraints also make sure that small weights are set to zero during training, and limits the search space by constraining the hypothesis space.

4.2.2 The BpTower algorithm

BpTower neural networks are included in GYAN phase 1 because of their similarity to cascade correlation networks, however they employ a different approach of building the network. For example, the BpTower algorithm has a distinct advantage for (decompositional) rule extraction in Phase 3, *i.e.* hidden units in a BpTower network is trained with a Heaviside activation function. Whereas hidden units in a cascade correlation network is trained with the sigmoidal activation function bounded between zero and one. As a result, a decompositional rule extraction technique approximates sigmoidal activation function with a Heaviside function while extracting rules from cascade corre-

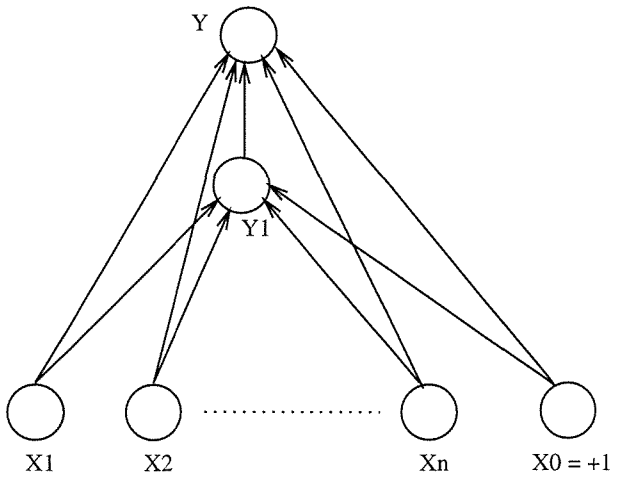


Figure 4.4: A feedforward network constructed by the BpTower algorithm.

lation networks, and may produce rules that fail to adequately describe the network.

The Tower algorithm [Gallant, 1990] employs feedforward single-cell learning to build a tower of cells, where each cell sees the original inputs and the single cell immediately below. The BpTower algorithm [Hayward, 1999] is the same as the Tower algorithm except that BpTower uses *gradient descent* to induce half-spaces rather than the *pocket algorithm with ratchet* [Gallant, 1990]. The weight-decay term discussed above is also implemented in the training algorithm.

The initial network starts with an input layer and an output node. BpTower constructs the network by initially training the output node on activations from all the input nodes to approximating the target function. When training stagnates, the node is inserted with connections from the input nodes, and the incoming weights are frozen. When the node is added to the network, its sigmoid activation function is replaced by a Heaviside function. If the network with this added node gives an improved performance and the termination criteria is not matched, then a new node is added with connections from the input nodes plus the recently trained node(s). Training of the newly constructed

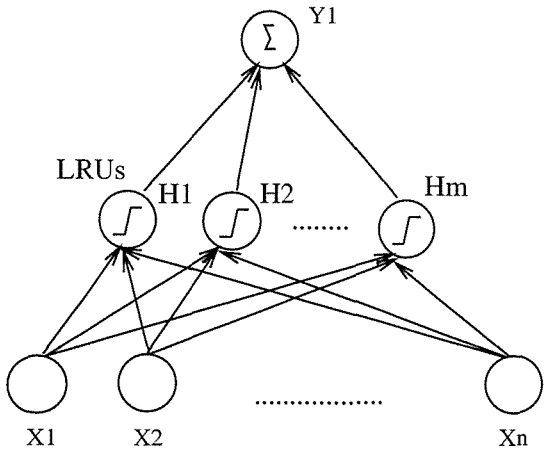


Figure 4.5: A CEBPN constructed by local basis function nodes

network is then started. The process is repeated until an acceptable overall network is generated [Figure 4.4]. The termination criteria are: the root mean square error (RMSE) on the validation (test) set reaches the minimum required error; or the RMS error on the validation (test) set starts to increase; or the maximum permissible number of hidden nodes has been added.

4.2.3 The constrained error backpropagation algorithm

Constrained error backpropagation networks (CEBPNs) [Andrews and Geva, 1996] are selected in GYAN phase 1 as they represent a special class of feed-forward networks that utilize *local functions* in the construction of hidden layers. The CEBPN performs function approximation and classification very similar to radial basis function (RBF) networks [Tresp *et al.*, 1993]. Lapedes and Faber [1987] describe a method to construct locally responsive units using pairs of axis parallel sigmoids. Here local response unit means that a node only responds to a selective range of possible values of input variables. Later, Geva and Sitte [1992] proposed a parameterization and training scheme for networks composed of such sigmoid based hidden nodes.

The CEBPN consists of an input layer, an output layer and a hidden layer of local basis function nodes (Figure 4.5). The hidden nodes are sigmoidals forming locally responsive units, rather than Gaussian units, that have the

effect of partitioning the training data into a set of disjoint regions, each region being represented by a single hidden layer node. A linear combination of such hidden layer nodes is able to approximate the target concepts of a problem domain. A set of ridges forms a local unit (hidden layer node), one ridge for each dimension of the input space. A ridge produces appreciable output (the thresholded sum of the activations of the sigmoids) only if the value presented as input lies within the active range of the ridge. For each ridge (in the i^{th} dimension), the axis parallel sigmoids are parameterized according to the centre, breadth and edge steepness of each ridge. The pair of sigmoids is given by the following equations [Andrews and Geva, 1994]:

$$U_i^+ = \frac{1}{1 + e^{-(x_i - c_i + b_i)k_i}}$$

$$U_i^- = \frac{1}{1 + e^{-(x_i - c_i - b_i)k_i}}$$

where x is the input vector, c, b and k are the centre, breadth and edge steepness of each ridge respectively. The local response region (a ridge parallel to the axis in the i^{th} dimension) is created by subtracting the value of one sigmoid from the other. The activation of this sigmoid is given by:

$$V = \frac{1}{1 + e^{-(\sum(U_i^+ - U_i^- - B)K}}$$

where B is the dimensionality of the input domain and K is a constant, set in the range 4 to 8. An incremental constructive training algorithm is used to adjust the parameters of the sigmoids (centre, breadth and edge steepness defining the local response units) by gradient descent. During training the output weight is held constant at a value such that the hidden nodes are prevented from overlapping, *i.e.* no more than one node contributes appreciably to the network output. This constraint further facilitates direct decompilation of each hidden node into an equivalent symbolic rule.

4.3 Phase2: Pruning

The constructive neural learning algorithms add new nodes with full connectivity to the existent ANN. After the trained network converges, links between

the hidden/output nodes and irrelevant input nodes may still carry non-zero weights. These superfluous non-zero weights, though usually small, make the neural representation difficult to interpret. In order to derive a concise set of symbolic rules from trained ANNs, one of the option is to eliminate all the irrelevant nodes and links from the network before the rules are extracted.

The precise steps to prune the irrelevant links and redundant nodes from the trained ANNs are summarized in Figure 4.6. There are two basic concerns involved in a pruning process:

- When should the pruning start?
  The algorithm [Figure 4.6] starts pruning when the minimum RMS error on the training and validation sets has been reached for the network.

- How many connections should be removed in each pruning step?
  The goal is to reduce the input space in order to apply a rule extraction method successfully in a smaller solution space, resulting in the generation of a more compact set of rules. A heuristically guided decompositional pruning process (inspired by the *MofN* algorithm [Towell and Shavlik, 1993]) has been adopted that tests clusters (the weighted links) at each pruning step, labels them if they are irrelevant, and eventually deletes all of them.

An effective means to prune an ANN is to employ a clustering algorithm on weight matrices to perform an appropriate reduction. A clustering algorithm picks out the distinctive subsets of weights embedded in a high dimensional space and selects lower dimensional weight matrices. Reduced dimension now enables the identification of a rule set which may otherwise remain hidden in high dimensional space. The pruning algorithm in Figure 4.6 applies a *metric distance* based clustering method to find the *best n-way partition* such that the metric difference among all the elements/weights in a cluster is less than the *distance measure* provided by the user [Hartigan, 1975]. Clustering starts by assuming each weighted link to be a distinct cluster and successively merging

1. **For** each non-input neuron in the network:
   1.1 Group the network's links of similar weights in clusters;
   1.2 **For** each cluster:
      1.2.1 Set the weight of each element to the average weight of that cluster;
      1.2.2 Test the cluster's magnitude against the bias weight;
      - **If** bias > cluster's magnitude **then** the cluster is marked as unnecessary;

   1.3 Sequentially present all the training examples to the network.
      1.3.1 **For** each cluster:
      - Set all the relevant weights to zero;
      - Label the cluster as unnecessary if there is no qualitative change in the network's prediction and the accuracy of the network is maintained high;
      - Adjust the relevant weights to previous values.

2. **For** each non-output neuron n_i in the network:
   2.1 Delete the links labeled as irrelevant;
   2.2 **If** all the connections of n_i are labeled as irrelevant **then** delete n_i;
   2.3 Remove n_i if there are no output links from it and n_i is a hidden node.
3. Train the remaining nodes and links for a few epochs.

Figure 4.6: The pruning algorithm

the two nearest clusters according to the distance metric. The difference in average magnitudes of each pair of clusters should be more than the set distance metric.

Threshold pruning is applied to eliminate redundant links and possibly input or hidden nodes from the trained neural network architecture while preserving the predictive accuracy of the trained neural network. The idea is that links with sufficiently low weights ($(|\sum weights| < Bias)$) are not decisive for a neuron's activation state, and are not contributing towards the classification of any of the examples. Consequently, low weighted links are not retained for symbolic rules, and are labeled as irrelevant, (step 1.2 in Figure 4.6). This method is fast and provides a first cut elimination of unwanted nodes and links. There is a modification in the threshold pruning for the cascade type of networks (cascade-correlation and BpTower). Each node in the cascade type of networks tries to learn a part of the target concepts (acting as a high-order

feature detector) [Fahlman and Lebiere, 1990]. If a hidden node is inserted in the network to learn a negative concept (usually indicated by a large negative weight to the output node), the corresponding input space should be identified. As a result clusters failing the test, ($|\sum weights| < Bias$), are considered to be relevant for labeling, *i.e.* they are kept in the network.

Assuming the sparse-coded input space (boolean variables according to each value of the attribute), there is a node for each attribute-value, and a weight is generated corresponding to each node. There is a probability that more than one (weight) values of an attribute are grouped in the same cluster. To ensure that only a single value of an attribute is represented in the cluster in any instance, all the samples are presented to the network for testing the effectiveness of clusters (step 1.3 in Figure 4.6). To avoid over-pruning and to maintain predictive accuracy of the underlying network, the two-fold provisions are: (1) the results from steps 1.2 and 1.3 in Figure 4.6 are merged to assure that only the clusters that are irrelevant (agreement from both steps) are deleted; and (2) if the classification accuracy of the remaining network is accepted then only is the cluster removed from the network. Furthermore, whenever an intermediate node loses all of its input links due to link deletion, this node too is removed from the network.

Only weights that do not effect the network's performance are removed in the pruning process. The network maintains high accuracy during elimination of clusters. As a result, only a few epochs are needed to train the remaining network once all the redundant nodes and links are eliminated. The pruned network is further trained using the quick-prop algorithm [Fahlman, 1988]. Quick-propagation is one of the most effective and widely used adaptive learning rules and simple in computation. The update rule for each weight is:

$$\Delta w(t) = \frac{S(t)}{S(t-1) - S(t)} \Delta w(t-1)$$

where S(t) and S(t-1) are the current and previous values of the summed gradient information (slope), $\delta E/\delta w$, over all the patterns in the training set.

Quick-propagation uses a single global parameter, the ε parameter, and a set of conditions to increase or decrease the amount of weight at each computation depending on the sign and weight difference for the current slope and previous slope.

Finally the reduced set of trained weights of the remaining links in the network, the subset of the training examples (consisting of the remaining attributes) and an additional set of instances (query instances) are recorded for the next phase.

4.4 Phase3: Rule extraction

The next step is to interpret the knowledge embedded in trained ANNs as symbolic rules. Many successful rule-extraction techniques represent the extracted knowledge in a propositional attribute-value language. The idea in GYAN is to incorporate existing propositional rule-extraction techniques and to enhance the expressiveness of the generated rules. The increase in expressiveness is due to the introduction of universally quantified variables, terms, and predicates in the extracted symbolic rules.

4.4.1 Propositional rule-extraction techniques

The GYAN methodology (Figure 4.1) includes the pedagogical rule-extraction technique *RuleVI* [Hayward *et al.*, 1997], and the decompositional techniques *LAP* [Hayward *et al.*, 1996] and *RULEX* [Andrews and Geva, 1994] in its environment to extract propositional normal form expressions. These rule extraction techniques are selected in GYAN phase 3 because they embrace a broad spectrum of ideas on rule extraction such as repeatedly querying the trained network and examining the response (*RuleVI*), or applying a search and test strategy to determine the condition under which a given hidden or output unit will be active (*LAP*), or directly encoding the weight matrices for each local basis function unit (*RULEX*). Use of these decompositional and pedagogical

rule extraction algorithms demonstrates the feasibility of applying the GYAN methodology to a diverse range of problem domains. If one technique fails (inaccurate output) for a problem domain, others can be applied successfully. Also the use of various techniques assures that the algorithm in phase 4 can generally be used to enhance the expressiveness of propositional rules - generated in any format (each of these techniques produces output in different form).

The pedagogical rule extraction technique - *RuleVI*

The *RuleVI* [Hayward *et al.*, 1997] pedagogical rule extraction technique is used to obtain the DNF expression (disjunction of conjunctive expressions) equivalent to the whole network trained for the given problem. The motif of the *RuleVI* [Figure 4.7] technique is that a conjunctive rule holds only when all the antecedents in the rule are true, and hence by changing the truth value of one of the antecedents the consequent of the rule changes. *RuleVI* generates a rule set by repeatedly changing antecedents of the training instances, querying the trained ANN, and examining the network's response. *RuleVI* utilizes *VI analysis* [Thrun, 1995] to create partially specified instances to make a proposed rule more general.

The *RuleVI* algorithm proceeds by initializing the DNF expressions for each target class as empty. An *Example oracle* generates the query instances. Each query instance is classified by the network to obtain a prediction. The query instance is then iteratively altered by a change in each of its dimensions. The amended instance together with the classification obtained by the network for the original instance is passed to the *Subset oracle*. The *Subset oracle* returns false if there is a disagreement between the classification of the amended instance and the classification of the original instance. The *Subset oracle* utilizes the trained ANN as an expert in classifying amended instances to test for agreement between them. If the *Subset oracle* returns false than the attribute's original value is added as a conjunctive term to the rule being

```
/* Initialize rules for each class */
For each class c
    R_c := NULL
Repeat
    e := Example()
    c := Classify()
    If e not classified by R_c Then
        r := empty rule
        Utilize VIA to form a set of amended instances (t) from e
          such that t is a vector of ranges rather than values
        For each input t_i
            u := t but with t_i a range
            If Subset(u) = false Then r := r AND t_i
            Else t := u
        R_c := R_c OR r
Until training instances are exhausted
```

Figure 4.7: The *RuleVI* Algorithm [Hayward *et al.*, 1997]

constructed indicating that this value of the attribute is contributing towards the network's output.

Importantly, inputs for this algorithm is restricted to binary values, either 0 or 1. The application of *VI analysis* requires the assignment of a valid interval (range) to each unit in the network in which the unit can fire. To determine a unit's output, the range is calculated instead of calculating a value [Hayward *et al.*, 1997]. The query instances are generated by amending each training instance such that each input dimension is a valid interval (min = 0, max = 1) in the instance. For example, if an instance contains n values, and each value is chosen from one of m valid intervals, then the instance will be altered and checked $n \times m$ times to generate a rule. The ranges are propagated through the network and the output vector, which will be a vector of ranges rather than values, can be tested to generate a valid rule. This algorithm is an improved version of the algorithm *RuleNeg* [Pop *et al.*, 1995] in which query instances are created from the original instances with each input dimension logically negated.

Each extracted conjunctive expression contains only one value per attribute and any $1 \ldots n$ number of attributes (values), where n is the total number of

attributes. *RuleVI* is designed to generate rules for each input pattern, and is able to extract rules covered by all the training patterns in the problem domain. The expressions are of the form:

$$y_1 \Leftarrow x_{11} \wedge x_{24} \wedge x_{36}$$

where x_i is an attribute, x_{ij} is a value for ith attribute and y_k is the target class.

The decompositional rule extraction technique - *LAP*

The *LAP* [Hayward *et al.*, 1996] decompositional rule extraction technique is used to obtain the disjunction of CNF expressions equivalent to each individual node in the trained ANN. The *LAP* rule extraction technique [Figure 4.8] decomposes the network into a collection of networks, and extracts a set of rules describing each constituent network. The core idea in *LAP* is to isolate the necessary dependencies between inputs and each non-input node in the network, and form a symbolic representation for each node. To achieve this the sigmoid activation function employed during training is approximated by a Heaviside activation function.

The *LAP* algorithm assumes that data presented to the network has been sparsely coded and uses this information to reduce the search space when identifying rules. The algorithm also assumes that an attribute domain will have a minimum of two values; if an attribute has a single value then it produces the complement of the values itself. The algorithm starts by constructing a set of weight vectors (W_1, W_2, ..., W_N), where W_i denotes the weight vector for the i^{th} sparse-coded attribute. The algorithm proceeds by testing the sum of the maximum-valued weights from each vector against the bias for each non-input node. If the activation for the output node is higher than the bias, then *LAP* iteratively constructs a separate case of weight vector for each attribute excluding the largest weight from each vector. There will be N separate cases formed for N attributes. The process is repeated until the node fails to fire *i.e.* no possible combination of weights is left in the set that can cause a high

```
Function LAP (W_1, W_2, ..., W_N) : Boolean is
  - Function enumerates recursively all minimal vectors, returns true
  - if max(W_1) + max(W_2) + ... + max(W_N) > unit threshold

  Boolean belowThresh, minimal;
  belowThresh := (max(W_1) + ... + max(W_N) > unit threshold)
  If belowThresh Then minimal := true
     For i := 1 to N
           If |W_i| > 1 Then
              tmp := W_i
              W_i := W_i - max(W_i)
              minimal := LAP (W_1, W_2, ..., W_N) AND minimal
              W_i := tmp
     If minimal Then
           save(W_1, W_2, ..., W_N)
  Return belowThresh
```

Figure 4.8: The *LAP* Algorithm [Hayward *et al.*, 1996]

activation of the output node. The inputs corresponding to the weights that cause the node to produce a higher output than the bias, then form the basis for CNF expressions.

Each extracted CNF expression contains multiple values per attribute (as a disjunction) and any $1 \ldots n$ number of attributes in conjunction, where n is the total number of attributes. The complete rule set is represented by the disjunction of all such CNF expressions. Each CNF expression is of the form:

$$y_1 \Leftarrow (x_{11} \vee x_{13}) \wedge x_{24} \wedge (x_{31} \vee x_{33} \vee x_{36})$$

where x_i is an attribute, x_{ij} is a value for ith attribute and y_k is the target class.

The decompositional rule extraction technique - *RULEX*

Unlike other decompositional methods, *RULEX* [Andrews and Geva, 1994] does not employ an *exhaustive search and test strategy* to extract rule sets to describe the behavior of trained ANNs. One of the salient features of *RULEX* is that the decision planes are parallel to the attribute axes, that facilitate rule extraction. The technique is designed to exploit the manner of construc-

tion and consequent behavior of a particular type of multi-layer perceptron, CEBPNs. In this type of locally responsive network, the function approximation and classification is performed by mapping a local region of input space to a hidden node. Thus each hidden node responds to a localized region of input space. The *RULEX* rule-extraction technique directly maps these localized regions into rules when the network uses locally responsive units during training. The basic idea behind *RULEX* is that if the activation of a ridge lies between its minimum and maximum value, the input value corresponding to the ridge becomes an antecedent in the rule. The cross-section of a local response ridge gives the range of input values that are sufficient to cause output of the ridge greater than a certain threshold. A ridge is formed by a pair of sigmoids for each dimension of the input domain. A hidden node is formed by a number of ridges equivalent to the dimensionality of the input domain. Each individual hidden node of the trained CEBPN is decompiled into rules of the form:

$$\text{If} \quad \forall i, 1 \leq i \leq n : x_i^j \in \{x_{i_{lower}}^j \ldots x_{i_{upper}}^j\} \quad \text{Then} \quad [\text{the target class}].$$

Where $x_{i_{lower}}^j$ and $x_{i_{upper}}^j$ represent the lower and upper limit of activation of the i^{th} ridge in j^{th} hidden node. *RULEX* expresses the complete rule set as a disjunction of CNF expressions. Each CNF expression contains any $1 \ldots n$ number of attributes with multiple values per attribute (as disjunctions). The expressions are the same form as of *LAP*, except the negated antecedents are allowed to appear. *RULEX* is suitable for both continuous and discrete data. *RULEX* reduces the size of the extracted rule set by post-processing such as removing redundant antecedents and rules, and use of negation in antecedents.

4.4.2 Mapping of propositional rules to predicate rules

This section describes the process for the generation of predicate rules (with variables) from propositional rules (without variables). The motivation for preferring predicate rules over propositional rules is the greater expressiveness of the former. Predicate rules allow learning of general rules as well as learn-

ing of internal relationships among variables. Sometimes comprehensibility of the extracted propositional rule set from ANNs is poor. Consequently it becomes hard to understand the hundreds of rules extracted by propositional rule-extraction techniques. A means to generate fewer general rules that are equivalent of many more simple rules in propositional ground form is necessary. A possible step in this direction is to map the propositional rules into quantified rules in the form of generic predicates, and to automatically create a knowledge base that includes predicate rules with variables, facts and a type-hierarchy. This process is independent of any network architecture and can be applied to any set of propositional expressions. The objective is to establish a mapping from propositional expressions to predicate rules (a more general description) by automatic bottom-up processing utilizing Plotkin's lgg concept.

Features of the knowledge-representation formalism

The representation formalism of predicate rules with variables, facts and a type-hierarchy adopts conventions from logic programming and Prolog [Lloyd, 1987], and is accepted by the SHRUTI reasoning system [Shastri and Ajjanagadde, 1993] as the knowledge representation language. This formalism has the following features:

- Each rule is expressed in the form of: consequent $\Leftarrow$ antecedents. Disjunctive consequents in a rule are written as separate rules[1].

- Antecedent predicates are a sequence of predicates, using commas or explicit ANDs to concatenate them. A predicate is a predicate symbol starting with a lowercase letter followed by a number of terms. Definitions of predicates and terms are the same as those in first-order logic except that terms are *function free*. The explicit negation of predicates is allowed in describing the goal concepts to avoid 'negation-by-failure'.

[1]The current version of SHRUTI admits rules with multiple consequents. Moreover, rules can be evidential.

- A fact is an instantiated/ground predicate if all its predicate variables are instantiated.

- A variable is a sequence of alphabetic or numerical characters in which the first character is an uppercase letter. A constant is a sequence of lowercase alphabetic or numeric characters. There are some restrictions on the appearance of variables in rules: any variable occurring in multiple argument positions in antecedents of a rule must also appear in the consequent [Shastri and Ajjanagadde, 1993]; any variables occurring in antecedents of a rule must occur in the consequent of the rule [Hayward, 1999].

- There is a single-depth type-hierarchy corresponding to input space of an ANN, in which attributes (such as Color) are concepts, and their values (such as blue, red, yellow) are sub-concepts.

- Execution is controlled by forward chaining or backward chaining.

- A query is a question mark (?) followed by a predicate with attributes which are either constants or variables or existential variables (denoted by a wild card !) or combinational.

This type of format offers a wide range of desirable properties e.g., the representation of relational knowledge, a better understandability of hypotheses, etc.

The generalization algorithm

A rule set extracted from a trained ANN by rule-extraction techniques is usually represented as propositional ground form expressions. The generalization task can be formulated as the task of finding a generalized rule set represented in the subset language of first-order logic such that $KR^+ \models C_1^+ \wedge ... \wedge C_n^+$ and $KR^- \models C_1^- \wedge ... \wedge C_n^-$, where KR^+ and KR^- are knowledge representations that cover all positive (C_i^+) and negative (C_i^-) conjunctive or CNF expressions respectively. The process for generating predicate rules from the propositional

rules is summarized in Figure 4.9.

1. Search for a DNF expression or a disjunction of CNF expressions equivalent to the neural network.
2. Generate a single-depth type-hierarchy by input-space mapping, with attributes as concepts, and values as sub-concepts.
3. Perform a symbol mapping for predicates to convert each conjunctive expression into a ground fact (such as $Nodename\#_1\_\#_2$, $hidden1\_1$ or $output1\_2$, or simply $p\_1, p\_2, .., p\_n$).
4. Utilize the fact definitions to create specific clauses (clauses with constants, C_1,C_2,..,C_n).
5. **For** all specific clauses do
   - 5.1 Search for any two compatible clauses C_1 and C_2.
     Let $C_1 \equiv \{l_1, .., l_k\}$ and $C_2 \equiv \{m_1, .., m_k\}$
     where each l_i, m_i has same predicate and sign.
   - 5.2 **If** such a pair C_1 and C_2 exists do
     - 5.2.1 Determine a set of selections, $S(C_1, C_2) := \{(l_1, m_1), .., (l_k, m_k)\}$
     - 5.2.2 Compute a new word symbol to hold the two k-ary predicates
       $word_1 := Temp(l_1, .., l_k)$, $word_2 := Temp(m_1, .., m_k)$
     - 5.2.3 let $\theta_1 := \emptyset$, $\theta_2 := \emptyset$, $q_1 := word_1$ and $q_2 := word_2$
     - 5.2.4 **While** $q_1 \neq q_2$ do
       - Search arguments of q_1 and q_2
       - find $t_1 \in q_1$ and $t_2 \in q_2$ such that t_1 and t_2 are occurring at the same position in q_1 and q_2 and $t_1 \neq t_2$ or one of them is a variable.
       - Replace t_1 and t_2 with a new variable X whenever they occur in the same position of q_1 and q_2.
       - Let $\theta_1 := \theta_1 \cup \{t_1/X\}$, $\theta_2 := \theta_2 \cup \{t_2/X\}$
     - 5.2.5 A rule with predicates and variables is generated
       $(word_1 = q_1\sigma_1, word_2 = q_2\sigma_2)$
6. **Return** the knowledge representation consisting of rules in the subset language of first order logic, facts and a type-hierarchy.

Figure 4.9: Process to generate the formalism of predicate rules from propositional rules based on Plotkin's lgg concept.

The first task is to convert each conjunctive or CNF expression into a ground fact. Each conjunctive expression extracted from the *RuleVI* pedagogical rule-extraction technique contains only one value per attribute, resulting in one fact. But the CNF expressions extracted from the *LAP* and *RULEX* decompositional rule-extraction techniques may contain more that one value for an attribute. This type of expression needs to be transformed according to the *'term-rewriting rule of generalization'* (chapter2, section 2.1), as they

result in multiple facts.

Some *minimization procedures* are applied to remove the redundant facts or entities in facts. These are:

- The first reduction is *removal of duplicated instances of facts* while transforming the conjunctive expressions into facts. This situation arises when a conjunctive expression is a subset of another conjunctive expression or contains a part of another conjunctive expression.

- The second reduction rule involves the *removal of redundant entities in compatible facts* (same predicate symbol and sign). The two facts, having all the entities equal except one, are merged into a single fact, for example: $p(a \wedge b \wedge c)$ *and* $p(a \wedge b \wedge \neg c)$ results in the fact, $p(a \wedge b)$, or $p(a \wedge b \wedge \neg c)$ *and* $p(b \wedge c)$ results in the facts, $p(a \wedge b)$ and $p(b \wedge c)$.

- The third reduction is *removal of specific facts by more general ones.* According to this rule, the fact p_2 can be removed by the fact p_1 if all the entities in p_2 exist in p_1, such that $p_2 \subseteq p_1$. Consider a knowledge base contains: $p_1(a \wedge b)$ and $p_2(a \wedge b \wedge c)$; p_1 is sufficient to match a query containing the entities a, b, and c.

There are many ways to compute predicate symbols during the generalization process, similar to the *'counting arguments rule of generalization'*. One way is to just opt for any letter followed by a counter, to denote a new instance of a ground fact (such as $p_1, p_2, ..p_n$). Another way is to take the network's architecture into consideration, predicate symbols corresponding to the position of nodes in the network. For example, the generated predicate symbol could have the structure $hidden\#_1\_\#_2$ for turning the conjunctive expressions into facts representing hidden nodes, where $\#_1$ denotes the position of the node in the network and $\#_2$ denotes the occurrence of fact instances. Similarly output nodes will have the predicate symbol $output\#_1\_\#_2$. If there are multiple conjunctive expressions present in the rule set containing the same number of

terms occurring at the same positions and indicating the same consequent (the node's output high or low), then these expressions are mapped to the same predicate symbol. In other words, this particular predicate will have multiple instantiations. Further, the fact definitions are utilized to express specific rules. These specific rules can now be expressed as clauses (disjunction of literals) by applying the logical equivalence law, $P \Rightarrow Q \equiv \neg P \vee Q$.

Plotkin's [1971] *'θ-subsumption rule of generalization'* has been utilized to compute the mapping of literals of more specific clauses to literals of more general clauses. In computing the generalization of two clauses, literals must represent each possible mapping between the two clauses. This is done by forming a set of pairs of compatible literals[2] from the two clauses, in the same way as is done for Plotkin's concept of *selection* [Plotkin, 1971; Wrobel, 1996]. The set of selections of two clauses $C_1 = \{l_1, .., l_k\}$ and $C_2 = \{m_1, .., m_k\}$ is defined as:

$$S(C_1, C_2) := \{(l_i, m_j) | \forall l_i \in C_1 \wedge m_j \in C_2 \wedge compatible\}$$

The least general generalization (lgg) of two literals requires to be computed first, and hence the lgg of two terms (function free) for computing the least general generalization of two clauses. Given a set of all selections of literals from two clauses, the clause lgg can be reduced to literals lgg by forming a special word to generalize (Figure 4.9). Wrobel [1996] defines word as a term or a literal, these words can be subsumed by the Plotkin's [1971] word-lgg algorithm. The lgg of two clauses C_1 and C_2 is defined as:

$$lgg(C_1, C_2) = lgg(S(C_1, C_2)) = lgg(Temp(l_1, .., l_k), Temp(m_1, .., m_k))$$

$$lgg(l_1, m_1) = p(lgg(t_1, s_1), .., (t_n, s_n))$$

A substitution $\theta = \{X/t_1, X/t_2\}$ uniquely maps two terms to a variable X in compatible predicates by replacing all occurrences of t_1 and t_2 with the variable X, whenever they occur together in the same position. This ensures that

[2]Two literals are said to be compatible iff they have the same predicate symbol and sign.

θ is the proper substitution of t_1 and t_2. The size of the set of selections of two clauses C_1, C_2 can be at most $i \times j$, where i is the number of literals in C_1 and j is the number of literals in C_2. In general the resulting lgg of two clauses contains a maximum of $i \times j$ literals, many of which may be redundant and can be reduced by applying Plotkin's equivalence property (chapter 2, section 2.1).

The lgg of two incompatible literals is undefined [Plotkin, 1971]. If there is a rule (with constants) left alone in the original set that does not have a pair with which to generalize this rule, is not reduced and just mapped in the appropriate format.

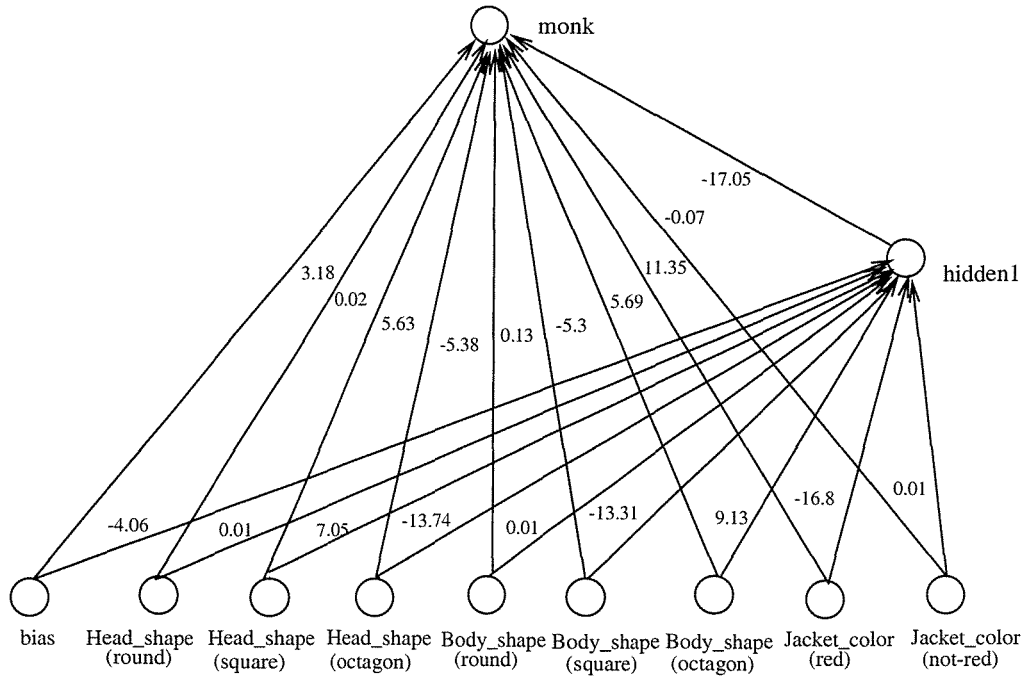


Figure 4.10: A pruned cascade correlation network solution of the Monk1 data set.

An example

A simple example of Monk1 (a detailed explanation is in chapter 5, section 5.2.3) will suffice for illustrating the rule generalization process. The decision rule for membership of the target class (*i.e.* a monk) is: (1) *Head_shape* = *Body_shape*, or (2) *Jacket_color* = *red*. The pedagogical rule-extraction algorithm *RuleVI* is applied to the pruned and trained ANN solution of the

Monk1 data set (Figure 4.10) to extract an equivalent DNF expression. After the pruning process, the input space to the ANN is: *Head_shape* $\in$ {*round, square, octagon*}, *Body_shape* $\in$ {*round, square, octagon*}, *and Jacket_color* $\in$ {*red, not-red*}. The DNF expression obtained by *RuleVI* for the *output node having high output* is:

1. Head_shape = round $\wedge$ Body_shape = round
2. Head_shape = square $\wedge$ Body_shape = square
3. Head_shape = octagon $\wedge$ Body_shape = octagon
4. Jacket_color = red

The extracted DNF expression indicating the *low output* for the output node is:

5. Head_shape = round $\wedge$ Body_shape = square
6. Head_shape = round $\wedge$ Body_shape = octagon
7. Head_shape = square $\wedge$ Body_shape = round
8. Head_shape = square $\wedge$ Body_shape = octagon
9. Head_shape = octagon $\wedge$ Body_shape = round
10. Head_shape = octagon $\wedge$ Body_shape = square

Each conjunctive expression can be expressed as a ground fact by adopting a predicate symbols scheme *i.e.* by labeling each conjunctive expression as the output node name (monk) followed by an instance number. As the first three expressions have the same number of arguments and the ith argument in each expression should refer to the same argument, they are mapped to the same predicate symbol, resulting in the following facts: *monk1(round, round), monk1(square, square), and monk1(octagon,octagon).* The fourth expression is inferred as *monk2(red).* Note that the expressions 5 to 10 have the same number of arguments as conjunctive expressions 1, 2 and 3, but they are indicating a different category (low output). Consequently, these expressions are mapped to a new predicate symbol, resulting in the following facts: *monk3(round, square), monk3(round, octagon), monk3(square, round), monk3(square, oc-*

tagon), monk3(octagon, round), and monk3(octagon, square).

A concept definition for the output node (as the consequent of rules) is formed by collecting dependencies among attributes (associated within facts) *i.e. monk(Head_shape, Body_shape, Jacket_color).* While mapping each ground predicate to a rule, if a fact only contains a subset of these attributes (Head_shape, Body_shape, Jacket_color) then the consequent predicate is filled with variables in the missing arguments positions. The unique-name and domain-closure [Brachman *et al.*, 1989] assumptions are adopted, which asserts that the domain of discourse includes precisely the attributes and their values, explicitly mentioned in the problem domain. Each attribute is presented by a unique variable. In the Monk1 example, X denotes the Head_shape attribute, Y denotes the Body_shape attribute and Z denotes the Jacket_color attribute. In other words, the constraint is imposed on rules that the entity bound to the variable X must be a subconcept of Head_shape for the inference to be valid. The specific inference rules including the ground facts are:

1. monk(round, round, Z) $\Leftarrow$ monk1(round, round)
2. monk(square, square, Z) $\Leftarrow$ monk1(square, square)
3. monk(octagon,octagon, Z) $\Leftarrow$ monk1(octagon,octagon)
4. monk(X, Y, red) $\Leftarrow$ monk2(red)
5. $\neg$monk(round, square, Z) $\Leftarrow$ monk3(round, square)
6. $\neg$monk(round, octagon, Z) $\Leftarrow$ monk3(round, octagon)
7. $\neg$monk(square, round, Z) $\Leftarrow$ monk3(square, round)
8. $\neg$monk(square, octagon, Z) $\Leftarrow$ monk3(square, octagon)
9. $\neg$monk(octagon, round, Z) $\Leftarrow$ monk3(octagon, round)
10. $\neg$monk(octagon, square, Z) $\Rightarrow$ monk3(octagon, square)

Now the remaining task is to discover dependencies among arguments, introduce variables in rules based on the dependencies, and generalize them. This is much like the process of finding the least general generalization of two clauses. The algorithm discussed in Figure 4.9 iterates over all the rules to

find two compatible rules. The algorithm first selects the two *compatible rules* 1 & 2 (same predicate symbols and signs). On applying the logical equivalence law, $P \Rightarrow Q \equiv \neg P \vee Q$, the rules 1 & 2 are transformed into:

1. ¬monk1(round, round) ∨ monk(round, round, Z)
2. ¬monk1(square, square) ∨ monk(square, square, Z)

Considering two choices for each antecedent, the set of selections of two rules contains a maximum of 2^n literals. These two clauses have two selections with consequent predicate, namely:

¬monk1(round, round), ¬monk1(square, square)
monk(round, round, Z), monk(square, square, Z)

A new word symbol *Temp* is utilized to form the two k-ary predicates with its selections:

1. Temp(¬monk1(round,round), monk(round,round,Z))
2. Temp(¬monk1(square,square), monk(square,square,Z))

The process of θ-subsumption proceeds with the following steps:

1. Temp(¬monk1(X,round), monk(X,round,Z))
2. Temp(¬monk1(X,square), monk(X,square,Z))

1. Temp(¬monk1(X,X), monk(X,X,Z))
2. Temp(¬monk1(X,X), monk(X,X,Z))

resulting in the inference rule:

• monk(X, X, Z) ⇐ monk1(X, X) with θ = [X/round] or [X/square]

This lgg rule is further θ-subsumed with rule number 3, which finally results in the following inference rule since there is no more rules compatible with this rule:

1. ∀ X,Z monk(X,X,Z) ⇐ monk1(X,X)

Similarly, the algorithm finds the lgg rule for the rules 5 to 10. A new word symbol *Temp* is utilized to form the two k-ary predicates to hold the set of selections generated from rule 5 and 6:

1. Temp(¬monk3(round,square),¬monk(round,square,Z))
2. Temp(¬monk3(round,octagon),¬monk(round,octagon,Z))

The θ-subsumption proceeds with the following steps:

1. Temp(¬monk3(round,Y),¬monk(round,Y,Z))
2. Temp(¬monk3(round,Y),¬monk(round,Y,Z))

resulting in the inference rule:

- ¬monk(round,Y,Z) ⇐ monk3(round,Y) with θ = [Y/square] or [Y/octagon]

This lgg rule is further θ-subsumed with rule number 7, and the same process of replacing the arguments with variables are applied. The steps are:

1. Temp(¬monk3(round,Y),¬monk(round,Y,Z))
2. Temp(¬monk3(square,round),¬monk(square,round,Z))

1. Temp(¬monk3(X,Y),¬monk(X,Y,Z))
2. Temp(¬monk3(X,round),¬monk(X,round,Z))

1. Temp(¬monk3(X,Y),¬monk(X,Y,Z))
2. Temp(¬monk3(X,Y),¬monk(X,Y,Z))

resulting in the inference rule:

- ¬monk(X,Y,Z) ⇐ monk3(X,Y) with θ = [X/round] or [X/square] and [Y/square]

This lgg rule is further θ-subsumed with the rest of the compatible rules, resulting in the following rule:

2. $\forall$ X,Y,Z $\neg$monk(X,Y,Z) $\Leftarrow$ monk3(X,Y)

For rule 4, the algorithm does not find any other compatible rule. This rule will therefore be:

3. $\forall$ X,Y,Z monk(X,Y,Z) $\Leftarrow$ (Z == red)

For the Monk1 universe of discourse, the following inference rules are generated:

1. $\forall$ X,Z monk(X,X,Z) $\Leftarrow$ monk1(X,X)
2. $\forall$ X,Y,Z $\neg$monk(X,Y,Z) $\Leftarrow$ monk3(X,Y)
3. $\forall$ X,Y,Z monk(X,Y,Z) $\Leftarrow$ (Z == red)

During mapping, it has been assured that each attribute is assigned to a unique variable and position in the target predicate such as, entity bound to the variable X must be a subconcept of Head_shape (round, square, octagon), Y:Body_shape (round, square, octagon) and Z: Jacket_color for the inference to be valid. These generated rules are able to capture the true learning objective of the Monk1 problem domain; that is, the higher order proposition that *(Head_shape = Body_shape)* (rule 1 & 2) rather than yielding each propositional rule such as *Head_shape = round and Body_shape= round* etc. These first-order rules make the non-localness explicit.

The SHRUTI implementation [Hayward, 1999] used for inferencing in this thesis does not realize soft rules (rules with constants or existing variables), the *'turning constants into variables rule of generalization'* (chapter2, section 2.1) is applied to map the constants of the rule to relevant variables. The generated rule is rewritten as: • $\forall$ X,Y,Z monk(X,Y,Z) $\Leftarrow$ monk2(Z) with the fact monk2(red) present in the knowledge base while reasoning.

4.5 Phase4: User interaction

The generated knowledge base can be interfaced with an inference engine that allows user interaction and enables greater explanatory capability. The inference process is activated when the internal knowledge base is operationally loaded and consultation begins. An efficient connectionist rule-based reasoner, SHRUTI, is interfaced to provide answers to user queries. The SHRUTI knowledge base reasoner has the ability to infer - to reach logical, consistent reproducible conclusions - from the facts by applying a rule set. In this phase, knowledge embedded in the ANNs is transformed into an equivalent inference network such as SHRUTI, where the input space corresponds to the concepts in the type-hierarchy, hidden nodes are mapped into intermediate predicates, and target or output nodes are mapped into consequent predicates at the highest level of the hierarchy.

4.5.1 The SHRUTI knowledge base system

Shastri and Ajjanagadde [1993] presented a connectionist reasoning system (SHRUTI) to efficiently enable dynamic inferencing based on the synchronous firing of constituent nodes in distinct phases. The primary concern of the SHRUTI model is to simulate the human cognitive ability to reflexively respond to a query by reasoning based on previously acquired knowledge. The knowledge in SHRUTI is represented as the connectionist implementation of rules and facts involving n-ary predicates and variables. The long term knowledge base (LTKB), regarded as rules and facts, is initially hardwired into the model. The rule-base reasoner can be integrated with a type-hierarchy to allow types, as well as instances, in the knowledge base and queries.

SHRUTI realizes the dynamic binding of fillers to predicate arguments by the synchronous firing of appropriate nodes. A dynamic binding between argument and constant nodes is represented by in-phase firing of both the nodes. In this model, rules are regarded as a systematic relationship between predicates

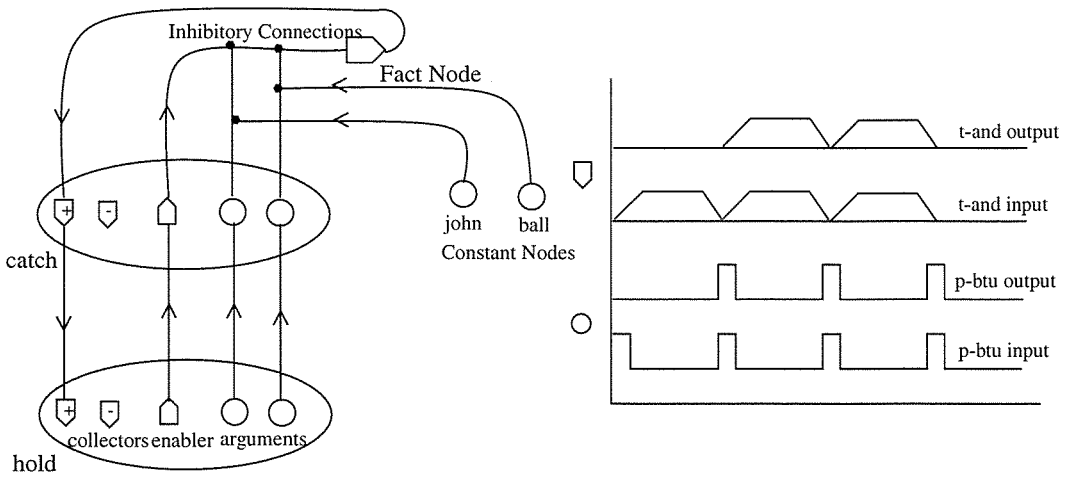


Figure 4.11: An example SHRUTI network encoding of the rule, *catch(x,y)* $\rightarrow$ *hold(x,y)*, and the fact, *catch(john, ball)*, after being posed the query, *?hold(john,ball)*. The oscillatory characteristics of nodes of types ρ-*btu* and τ-*and* are shown.

and are represented by appropriate links between the arguments of antecedents and consequent predicate nodes. The facts are regarded as instantiated predicates and represented as temporal pattern matching subnetworks, in which each argument in the predicate is permanently bound to the appropriate filler. The representation of a fact encodes the bindings pertaining to the fact in a manner that allows the system to rapidly recognize the bindings that match the encoded facts. A query is represented as a signal in the network that is propagated from one rule to the next.

A predicate is made of n-argument nodes, an enabler node and two collector nodes (positive and negative). During a reasoning episode, the constant values (fillers) are bound to the argument nodes of the predicate. Bindings are propagated by causing two nodes (an argument node and its filler) to fire in the same phase. The enabler node in the predicate propagates the query to the antecedent's predicates. The collector nodes become active when the predicate is instantiated. The incorporation of two collector nodes, positive and negative, allows for explicitly defined negated knowledge.

Basically, the oscillatory behavior of individual nodes in the network causes

the systematic propagation of information along existing inference paths to encode rules and locate supporting facts. When provided with input, each of the node types (represented by different shapes in Figure 4.11) in the predicate exhibits a distinct behavior. The enabler, predicate and fact nodes are encoded as *τ-and nodes* in the SHRUTI network. A *τ-and node* becomes active on receiving an uninterrupted pulse train for a length of time equal to the period of oscillatory activity, π. When active, the *τ-and node* continues to fire with an uninterrupted pulse train for a period of time, π. The *ρ-btu nodes* are used to encode arguments in the rules and concepts in the type-hierarchy in the SHRUTI network. A *ρ-btu node* produces a spike train synchronous (in phase) with the driving input on receiving a spike train during the previous period of oscillation.

SHRUTI supports forward and backward reasoning for inferencing. The inference process provides a way to extract the information that is implicit in a stated body of knowledge. The information encoded in the memory of the network is accessed by posing queries to the network. A query, which can be processed by SHRUTI in a reflexive manner, is the same as a predicate whose arguments are either bound to constants or existentially quantified. The query predicate and its arguments' bindings are specified when the query is posed to the system. To pose a query to the SHRUTI network, an uninterrupted pulse train is applied to the enabler node of the queried predicate for a period of time π. Each of the argument nodes in the predicate is caused to fire in-phase with the concept node in the is-a hierarchy involved in the query. During this period, the query is propagated to predicates connected through rules and simultaneously attempts to activate fact nodes associated with the predicates. When a fact node becomes active, it causes the appropriate collector node in the associated predicate to become active. The incorporation of two collector nodes, positive and negative, allows the system to respond to the query in four possible ways: true, if the queried predicate has the positive collector activated; false, if the queried predicate has the negative collector activated;

inconsistent with the knowledge base indicating ambiguity, if both the positive and negative collectors are activated; unknown, if both of the predicates remains inactive for some period of time. This means that the system does not make a closed world assumption when there are no facts or rules supporting the query.

The model can also be ported to parallel computing architectures where it has the unique feature that the response time for a query is dependent on the length of the inference path, rather than on the size of the knowledge base. There are some important restrictions in SHRUTI that affect the reasoning process:

- *Multiple occurrences of variables.* If there are multiple instances of the same variable in antecedent predicates, each instance must be consistently bound to the same constant during reasoning. SHRUTI can not solve a query like: $p(X, X))$, where the next inductive step is: $p(X, X) \Rightarrow q(Y, Z)$, because the system cannot differentiate between the two Xs of the query arguments and would still bind them to different arguments Y and Z. The query would be answered as true (which is incorrect) if the knowledge base has a supporting fact that instantiates two different arguments rather than the same. This is a consequence of the rule not being balanced. SHRUTI avoids this problem by imposing the restriction that such a variable can occur in the conclusion of a rule only. This is also handled by imposing the restriction on the rule sets that all the variables in the antecedents of a rule must appear in the consequent predicate of the rule in a backward reasoning system. There have been some advancements in the original SHRUTI model that allow the enhanced model [Shastri, 1999] to cope with all these situations with some relaxation of these restrictions.

- *n-ary function symbols.* n-ary function symbols are difficult to deal with in SHRUTI as they require dynamic construction of structured objects. A structured object is represented by a group of nodes, as it can not be

While (all the arguments in the query are valid) do
    If query parameter is a constant
        spread bottom-up activation in the type-hierarchy
    If query parameter is an existentially quantified variable
        spread bottom-up activation in the type-hierarchy
        spread top-down activation in the type-hierarchy
    If query parameter is a universally quantified variable
        /* no propagation in the type-hierarchy */
- Fire the enabler of the query predicate (activate for full period)

Repeat
- propagate activation in the rule-base (inference paths)
- check for facts matching
- if a fact is found matched, reverse propagate the activation to the query predicate (appropriate collector)

Until the query finds an answer or the inference process is exhausted

Figure 4.12: The algorithm to process a query

represented by a single node. Obviously, it is expensive and difficult to maintain and update such group of nodes [Hőlldobler, 1990].

- *Bindings between variables.* The SHRUTI version used in this thesis does not assign phases to variables. As a result, the SHRUTI inference system only guarantees to correctly respond to queries which contain at most one variable. For example, assume the facts: *gave(john,mary,book), gave(john,kate,ball)* are in a knowledge base. If the query: *?gave(john,mary,X)* is posed, the query returns true with X being *book*. Now if the query: *?gave(john,Y,X)* is posed, the query returns true with unknown bindings among *(mary,kate)* and *(book,ball)* (i.e. John gave Mary a ball or a book). Park *et al.* [1995] have proposed a solution to this problem by providing a mechanism that supports the assignment of unique phases to variables.

SHRUTI's query facility is much more simple and limited, than a relational database query system. Although the expressive power of such CSNs is limited, they allow efficient inference processes [Diederich, 1992]. The algorithm to process queries in SHRUTI is shown in Figure 4.12. SHRUTI provides an explanation of how a particular solution to a query has been reached using the steps in an inference and matching facts. Consider the Monk1 example from

the previous section. After the automated knowledge base is compiled with the SHRUTI reasoning system, SHRUTI allows the user to pose queries and also provides an explanation of why the resulting classification is made. For example, if the query *monk(square, square, not-red)* is posed to the knowledge base generated for the Monk1 data set (as explained in section 3.4.2), SHRUTI initiates and executes the appropriate rules for the given situations and returns the answer *true* with the explanation:

- monk(square,square,not-red) $\Leftarrow$ monk1(square, square)

In a similar fashion, the system is able to return *true* whenever the query has the same value for Head_shape and Body_shape, or the value red for Jacket_color. The system returned false for other instances where this condition is not met, such as *monk(round,square,not-red)*, which returns false with relevant facts.

4.6 Discussion: The Gyan methodology

The inductive methodology, GYAN, takes a set of examples (data set) $E = E^- \cup E^+$, and proposes a new set of descriptive statements (rule set) R, such that $R \vdash^m E$ where $\vdash^m$ indicates that entailment holds relative to the restrictive first-order logic mapping. While mapping, there should be no loss of information at each step of the process but rather an enhancement of the generalization accuracy and explanatory power of the model produced at each stage. There is a probability of existence $e \notin E$ such that $R \vdash^m e$ as a result of generalization. This means that e is not originally given in the example set E but is derivable from the generated rule-set R. There is also a probability of existence $e \in E$ such that $R \nvdash^m e$. This means that e is originally given in the example set E but is not derivable from the generated rule-set R. This is because the hypothesis learned by the ANN does not completely agree with training examples due to some kind of errors or noise in the data, or while approximating ANNs with symbolic rules, fidelity of rules is not 100%.

There are many issues such as the use of various algorithms in GYAN, the use of a connectionist system for rule processing, the algorithmic complexity of the GYAN methodology, etc., that need to be addressed. The ensuing discussion outlines some of them.

4.6.1 Use of various algorithms in Gyan

GYAN incorporates many algorithms in its framework to represent predicate rules from trained ANNs. This section discusses and justifies the use of various algorithms in GYAN.

Incremental neural learning algorithms

GYAN utilizes dynamically constructed ANNs to learn concepts from data. The feedforward multilayer networks have the ability to represent large data sets and are universal function approximators [Hornik *et al.*, 1989]; that is, for any given function there is a feedforward network capable of approximating the function arbitrarily closely. Incremental learning algorithms do not need *a priori* specification of the network architecture. They start from small networks and add nodes and interconnections as required until suitably chosen measures stop the process. In other words, incremental algorithms help to ease the problem of deciding the optimum size of the network.

During on-line (or batch) learning, weights are modified after each presentation of a training example (or an epoch). In case of non-constructive algorithms (multi-layer perceptrons), a good set of weights for recognizing a target may be distributed or forgotten when the weight adjustments are made for another target. But constructive algorithms like cascade correlation and BpTower overcome this problem by freezing its co-efficients, once a part of the network is learned. The other advantage of cascade correlation and BpTower algorithms is that dynamically building a network with localized hidden node representations that allows for lateral connections among hidden nodes. These lateral connections become part of the inference paths in a reasoning episode

when the resulting knowledge base (processed from the network using decompositional rule extraction techniques) is interfaced with an inference engine. This type of cascading of hidden units results in a network that can represent very strong non-linearities hidden inside data, and each hidden node acts as a feature detector [Fahlman and Lebiere, 1990].

Cascade vs BpTower algorithm

The cascade correlation algorithm is a top-down approach to building the network which stacks new hidden nodes in multiple layers with a fixed output layer. By contrast, the BpTower algorithm is a bottom-up approach to building the network that stacks each new hidden node as the output node in a progressively deepening network, in which the previous inserted node becomes the hidden node after being installed. The BpTower algorithm can only be applied to boolean decision problems because it approximates the activation function of each hidden node by a Heaviside activation function after being inserted in the network. The different modes of training in both the algorithms are easily illustrated by the weights produced in the trained ANNs especially by the weights corresponding to hidden nodes in the output layer (Figures 4.13).

The pruning algorithm

Most of the pruning algorithms to remove superfluous links from ANNs are *iterative in nature.* Examples are the Hagiwara algorithm [Hagiwara, 1993], OBS [Hassibi and Stork, 1994], and N2P2F [Setiono, 1997a; 1997b]. The iterative algorithms require retraining of the network after removing connections or nodes. The retrained network is then checked to see if any of its remaining nodes or connections meet the criteria for further removal. More often, the amount of computation necessary for retraining is much higher than that needed to train the originally fully connected network [Blassig, 1994].

An alternative approach of pruning is reduced-error pruning based on

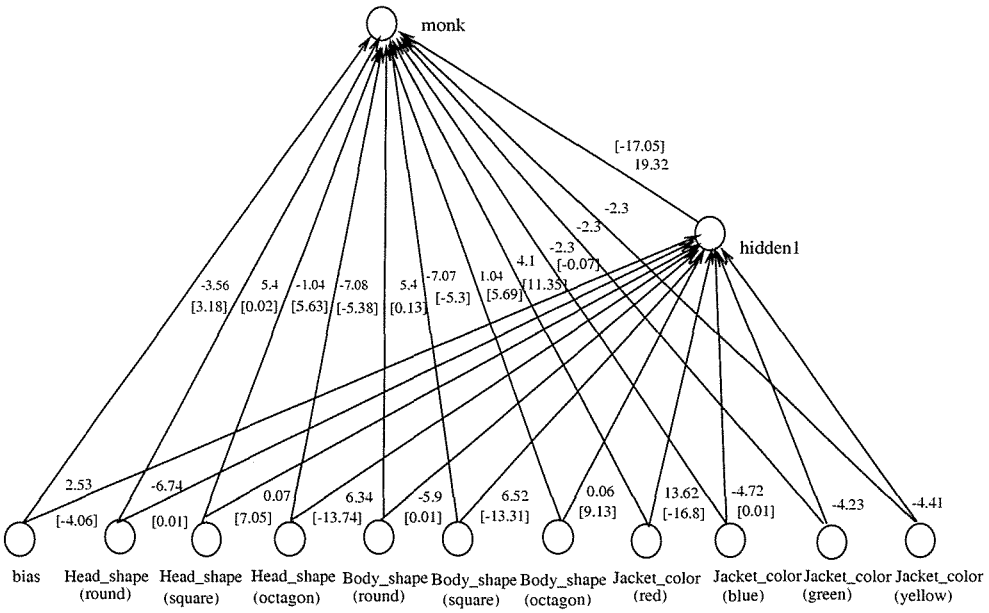


Figure 4.13: The pruned cascade correlation and the BpTower network solution of the Monk1 data set. The BpTower network solution requires two extra nodes as compared to the cascade network solution. The weights for the cascade network solution are shown in brackets.

the Minimum Description Length (MDL) Principle [Rissanen, 1983; Sporring, 1995]. In the MDL approach, possible theories (or neural networks) $\{T_i\}$ derived from data are characterised by their description length, the number of bits needed to encode both the theory and the data from which it was learned. Choosing the theory (or the neural network) T_i with minimum description length is equivalent to maximizing the probability $P_r(T_i|D)$ of T_i given the data [Quinlan, 1994]. Given the theory, the MDL principle can be viewed as a tradeoff between theory complexity and data prediction accuracy. Using the MDL principle for pruning of neural networks can be closely related to cross-validations schemes used in training of networks. Also quinalan [Quinlan, 1994] stated that sometimes MDL leads to poor choices among competing theories, as theories with larger categorical error rates tend to assign an unexpectedly high or low prior probability to the described (target) class.

The *distinguishing* feature of the pruning algorithm developed in this thesis (inspired by the *MofN* algorithm [Towell and Shavlik, 1993]) is the identifica-

tion of relevant connections from the input nodes to the hidden/output nodes based on the magnitude of their weights. The imposition of a few soft constraints upon the network during training encourages the pruning of nodes and links. The algorithm checks for acceptable predictive accuracy of the ANN during the labelling of irrelevant connections. Once all the superfluous connections are determined and the conditions are met, they are removed *all at once.* As a result, only a few epochs are needed to train the remaining links, and this is only done *once.* Pruning determines the important variables for rule-extraction from the trained ANN, resulting in a set of rules which has fewer antecedents and is maximally general.

The rules extracted by *LAP* are of high fidelity because their classification ability is equivalent to the network from which they are extracted. A shortcoming of *LAP* and other decompositional techniques is that these methods become cumbersome when the search space is too large (in terms of the number of attributes). As the number of dimensions increases, the number of possible combinations of attributes grows exponentially. Pruning allows the rule extraction methods to only consider the parts of the search space whose elements are involved in approximating the target concepts. In *LAP*, the dimensionality of the search space is exponential in the number of values of all attributes, leading to a substantial gain when all the concepts are not included in search. The rule sets extracted by *RuleVI* and other pedagogical methods only 'completely cover' the set of instances that are used to obtain them. In *RuleVI*, the number of amended instances to test the proposed rules is significantly reduced by removing attributes that do not have any impact on the target concept. Thus the complexity of such methods and of the extracted rules is decreased with an increase in rule quality (comprehensibility, predictive accuracy, etc). A projection to a space of lower input dimensionality translates an ANN into a compact rule set, that seems impossible to solve with a high number of input dimensions.

Extraction of predicate rules

Despite the fact that recursively defined predicates and infinite terms are not allowed in the generated knowledge representation, the predicate formalism is appropriate for real-life problems [Lavrac *et al.*, 1991]. The primary advantage of this representation is that implication is decidable [Bergadano and Gunetti, 1995, pp.35] (chapter 2, section 2.1, implication). The advantage of using ANNs to extract predicate rules from data is that the hypothesis is not expressed in first order logic format and concepts are not found in an infinite search space as compared to FOIL or other first-order logic learners. Using a first order language for the expression of hypotheses, many problems such as matching or testing for subsumption become computationally intractable and even the very notion of generality may acquire more than one meaning [Plotkin, 1970; Buntine, 1988].

It can be observed from the description of the generalisation algorithm that the length of the resulting lgg of two rules C_1, C_2 can be at most $((|C_1| \times |C_2|))$ literals. One generalised literal is included in the lgg for each *selection*, many of which may be *redundant* (section 2.1, equivalence and reduction). Since GYAN is concerned with explaining the ANNs comprehensively, the number of generated lgg literals is not much of a concern in GYAN. The generated DNF expression is constrained to a single-depth rule mapping (inputs to outputs) for each decompiled network (in case of decompositional techniques) or a whole network (in case of pedagogical techniques). The body of a rule contains at the most two literals (including consequent) to be generalised at any time. There may be more literals in a rule corresponding to hidden nodes if the rule-base is hierarchical (considering a cascade type of network architeture in which a node receives inputs from all input nodes and the previously inserted hidden nodes). But the rules for hidden nodes will already be generalised (as a result of using a bottom-up approach). In this generalisation process, the methodology is explained with regard to a single target concept (predicate). Since rules with different consequent predicates can never subsume each other,

the rules for each target concept are effectively generated independently of one another.

User interfacing - SHRUTI

Due to rapid growth of on-line systems, it is highly desirable that a system should help users to pose their queries in order to retrieve the information about the system that they are really interested in. GYAN opens up the possibility of interacting with data sets and neural networks by allowing the user to ask queries via the interface of a reasoning system such as SHRUTI. SHRUTI is a connectionist architecture that can encode millions of facts and rules involving n-ary predicates and variables, and performs inferences in a few milliseconds even if they are quite limited. SHRUTI is a biologically plausible system that has highly desirable properties such as fine-grain parallelism, fault tolerance, etc.

The use of SHRUTI brings the following advantages to GYAN: inference time is independent of the size of knowledge base; inferencing complexity is linear in the length of the inference path; there is support for parallel inferencing and mapping of a knowledge base onto parallel architectures. This user interfacing does not require the user to be a domain expert by allowing him/her to pose simple queries, even partially instantiated queries are allowed in principle. This type of queries and their responses (predictions, and fillers to the variables) can further be used to supplement data in input space if there are insufficiently represented regions.

4.6.2 Use of a connectionist system for rule processing

Critics of the connectionist movement (such as [Fodor and Z. W, 1988; Pinker and Prince, 1988] argue that connectionist systems can not be considered adequate for cognitive modeling. The idea, that knowledge is easily and naturally represented by and can be expressed in propositions that are sentence-like [Bechtel and Abrahamsen, 1991], seems to make the criticism stronger.

Pinker and Prince [1988] consider the lack of rules (presented in some kind of symbolic logical form) and 'variable binding'[3] to be a major shortcoming of connectionist systems. Fodor and Pylyshyn [1988] criticize that without symbolic representations it is not possible to account for certain aspects of human cognitive performances. They argue that connectionist systems (both localist and distributed) lack a combinatorial syntax and semantic [Fodor and Pylyshyn, 1988, pp. 24]. Although individual units or coalitions of units in a connectionist system may be interpreted semantically, they cannot be built into linguistic expressions and manipulated in accord with syntactic rules. The units are not symbols and the system is inadequate for its task of representation [Fodor and Pylyshyn, 1988, pp. 34].

They exemplify that a connectionist system will not be able to distinguish between 'John loves Mary' or 'Mary loves John'. The connectionist systems or nodes represent concepts rather than propositions, such that 'John loves Mary' is a distribution of activation over the set of nodes {John, loves, Mary} rather than the activation of a single node labeled 'John loves Mary' [Fodor and Pylyshyn, 1988, pp. 25]. They claim that the inference in a connectionist system is built in separately for each instance of conjunction rather than by means of a rule that utilizes variables to specify the syntactic relation of inclusion (due to 'variable binding' problem) [Bechtel and Abrahamsen, 1991, pp. 212]. For example, the unit A & B must be specifically linked to unit A if the inference from A & B to A is to be made, just as C & D had to be linked to unit C if the inference from C & D to C is to be made, there is no structural relation that holds between them [Fodor and Pylyshyn, 1988, pp. 16]. Symbolic systems usually employ variables so that rules can be applied to various entities. On this basis Fodor conclude that connectionist systems lack the requisite resources for cognition.

[3]Whenever a variable appears more than once in a rule, each instance must be bound to the same constant.

Bechtel and Abrahamsen [1991] put forward the idea that connectionism might provide detailed models of 'knowing how' based on Dreyfus and Dreyfus [1986] analysis that expert performance does not rely on propositionally encoded information, but rather on the ability to recognize situations as similar to previously encountered situations and to rely on what worked in those situations. Connectionist networks attempt to account for some or all of the cognitive performance that cognitive scientist have traditionally accounted for by means of symbolic models without explicitly employing propositions [Bechtel and Abrahamsen, 1991, pp. 151]. The challenge for connectionist networks is to perform the tasks with equal capabilities that are performed in symbolic systems by means of combinatorial structures of symbols strings which include variables.

The GYAN methodology adopts the *compatibilist* approach [Bechtel and Abrahamsen, 1991, pp. 238] that is, acquisition and implementation of the explicit rules in a connectionist network and maintain the crucial benefits accrue as a result of the connectionist implementation. The GYAN methodology applies distributed connectionist networks (ANNs) at the lower level. In rule extraction step, it introduces the variables and relations to the symbolic rules eliciting the knowledge embedded within trained ANNs. At the higher level reasoning, this generic and conceptual knowledge is represented in localist connectionist networks (CSNs) and provides user interface. The methodology results in a massively parallel, fault tolerant high-level learning and reasoning system by embedding symbol representations and structure-manipulating operations within a distributed, sub-symbolic architecture.

4.6.3 Algorithmic complexity of Gyan

Another issue to discuss is the algorithmic complexity of GYAN. The algorithmic complexity of GYAN depends upon the core algorithms used in different phases.

The steps in the pruning algorithm (phase 2) that consume most of the computational effort are clustering the network's links of similar weights, labeling/eliminating unnecessary clusters and training the remaining nodes and links for a few epochs. The initial clustring step requires $O(u \times l^2)$ time, where u is the number of non-input nodes in the trained network and l is the average numner of links received by a node. The cluster elimination step requires $O(n \times u \times l)$, where n is the number of training examples. Training of the remaining network requires only a few epochs that is much simpler than the initial training of the network. The reason is that only links that do not affect the network's performance are removed in the pruning process, and the network maintains high accuracy during elimination of clusters.

The *LAP* decompositional technique faces the problem that the search space to generate rules is exponential in the number of inputs to the network, although the heuristics involved do limit the size of the search for rules through weight space. Some more heuristics must be introduced to limit the search space as suggested in Chapter 7. The *RuleVI* pedagogical technique faces the similar problem of having a large search space to generate rules. Some suggestions of using suitable heuristics to obviate the need for enumerating all possible examples in the problem space is given in Chapter 7. Unlike other decompositional rule extrcation techniques, *RULEX* does not employ an exhaustive serach and test strategy, and does not face the problem of having a large search space. However as shown in Chapter 6, a significant reduction in the run-time for these algorithms can be acheived by removing the irrelevant attributes from pruning the network.

The generalisation algorithm used in phase 4 requires $O(l \times m^2)$, where l is the number of clauses according to the DNF expression equivalent to the trained neural network and m is the total number of attributes in the problem domain. However, application of the pruning algorithm in phase 2 significantly reduces the total number of attributes.

In comparison to rule extraction techniques where a logical expression is derived describing the overall behavior of a network, the GYAN methodology provides a much more detailed explanation of why a particular instance is classified as a member of a goal concept. While doing so GYAN pays a high price (in terms of computation) for the benefits of an improved explanation capability. However a judicious choice of heuristics can be used to extract reasonable solutions efficiently using GYAN. Also the use of reliable and general (portable and scalable) propositional learners can make GYAN more efficient.

Overall therefore, providing an explanation in terms of rules and facts involving generic predicates is a step forward in the symbolic representation of networks.

4.7 Chapter summary

The development of GYAN is motivated by a desire to comprehensively understand the decision process of an ANN, and to provide explanations to the user by interfacing the network's output with a knowledge base reasoner. The powerful advantage of ANNs, the ability to learn and generalise, is exploited to extract knowledge from a set of examples. Even though ANNs are only capable of encoding simple propositional data and they are not able to encode a wide variety of data such as rules involving variables, with the addition of the inductive generalisation step, the knowledge represented by the trained ANN is transformed into a representation consisting of rules with predicates, facts and a type-hierarchy. Furthermore, the methodology is extended to interact with the user, who is allowed to ask questions of restricted kinds, by interfacing the automated knowledge base with an inference engine after the learning process has been completed. By this means (1) a logical expression is derived that describes the overall behaviour of the network and (2) a detailed explanation (in terms of activated rules, facts and predicates in a reasoning

episode) is provided as to why a particular instance is being classified into a certain category. More generally, the methodology is able to use any type of machine learning classifier that generates propositional rules. Essentially, the qualitative knowledge representation ideas of symbolic systems are combined with the distributed computational advantages of connectionist models.

Chapter 5

Data analysis and representation

The previous chapter provides a detailed description of the GYAN methodology to enable data exploration, analysis, modeling, compression and querying at a much higher level of user-interaction. The first important task in GYAN (or any knowledge discovery process) is to prepare data for the relevant classifiers. The first section in this chapter discusses data representation issues - data preprocessing, data transformation and data distribution. The second section briefly introduces all the application domains that are used to analyze GYAN in the following chapter. The last section sheds light on the evaluation criteria and the reporting scheme that are used to present the outcome of GYAN in the following chapter. The main focus of this chapter is to introduce the design of the experimental analysis of GYAN.

5.1 Data representation

The proposed method GYAN is designed to analyze data, and to extract meaningful information from data patterns using neural networks. The first task in the methodology involves *data preparation - preprocessing, transformation and distribution* of data. It is important to determine whether data is amenable to the component machine learning classifiers in the methodology and to ensure

that adequate data exists and can be presented to a classifier.

A high quality *data preprocessing* is required to minimize ambiguity, error and randomness of data. The data representations for individual learning algorithms vary considerably. The task includes procedures to *transform the preprocessed data* to an equivalent representation required by the component programs. For example, rule-extraction techniques such as *LAP* and *RuleVI* require each pattern in the data sets to be presented as a *bit vector* rather than as a *continuous/multi valued* vector. Multiple-valued features are easily discretised into binary features. Though the continuous-valued features are treated differently, the values for these numeric features are divided into subranges and then discretised. Also the *resampling of data* is required to be included in the data representation task. The data set is usually subdivided into training, testing and validation sets. The data set should be partitioned into non-overlapping, fully or partially replicated portions.

5.1.1 Data preprocessing

The first task in GYAN, as in any knowledge discovery process, is to select the required data and to ensure the quality of selected data. Clean and well-understood data is a clear prerequisite for successful information extraction [Cabena *et al.*, 1997]. The preprocessing step also provides a better way to get acquainted with the data at hand. In this step, the attributes that are stored in data sets for identification purposes are removed from the data. Duplicated records are also deleted at this stage. A very important aspect of data preprocessing is the handling of noise and missing values. A deliberate decision has to be made whether to overlook these, or to delete them, or to replace them [Cios *et al.*, 1998]. Any deletion of data must be a conscious decision made after a thorough analysis of the consequences. Missing values can be handled by simply disregarding them; or omitting the corresponding records; or treating them as a valuable indication and including them as additional values in the attribute domain; or replacing them with the most likely

values [Perthold and Hand, 1999]. More details are provided in section 5.3 with each application domain, in which data preprocessing has been performed according to the specific requirement of the domain.

5.1.2 Data transformation

The goal of inductive learning is to formulate plausible general descriptions from the given data. In supervised learning, each data pattern in the instance space is described by a fixed set of attributes X_i (where $i = 1, \ldots, n$, and n is the number of attributes) and is tagged by the target attribute X_o. Each of the attributes can be either categorical or numerical. The domain of categorical attributes is discrete *i.e.* the values are from an unordered symbolic values set $\{v_{i1}, \ldots, v_{il}\}$, where l is the number of possible values of attribute X_i. The domain of numeric attributes is given by a numeric interval $[v_{min_i}, \ldots, v_{max_i}]$, which is considered continuous for the purpose of modeling.

Discretisation of data

Real world problems normally contain both numeric and discrete data. Many inductive learning algorithms (in particular rule-extraction techniques such as *LAP*, *RuleVI*, etc.) can only handle discrete data. Before running these algorithms on a data set that includes continuous-valued attributes, discretisation is necessary. This is a process that quantizes the numeric data $[min_i, \ldots, max_i]$ into a number of intervals, and maps each interval to a discrete value/symbol. The continuous values falling in the discretised region are replaced by the categorical value. Determining the intervals is in itself a practically useful exercise. The most frequently used discretisation methods are *equal width intervals, equal instance population, one dimensional clustering, etc* [Kerber, 1992; Fayyad and Irani, 1993]. These methods are easy to implement, but require the user to specify the number of intervals [Liu and Tan, 1995]. The ultimate goal is to discretise in such a way that the resulting model performs best on the modified data.

The GYAN methodology determines quantization intervals based on the *analysis of an accumulated frequency graph.* All the attributes (to be discretised) are first sorted in an ascending order. The accumulated frequency (accumulative number of occurrences of the specific data point) is then calculated for each instance in the data set. A graph of accumulated frequencies is drawn for each attribute, and the intervals are determined according to the slopes. The discretisation process constructs an abstraction space over the continuous attributes. Learning in this new abstraction space has several advantages. First, it allows for effective feature construction. Second, dependence analysis between continuous and nominal attributes can be performed. Finally, discretisation results in substantial speed-up for the inductive process, *i.e.* it cuts down the computational cost of the learning task [Catlett, 1991].

For the sparse-coding representation, each value of the discrete (categorical) attribute with n possible values is represented by an n-bit binary string, with only one bit carrying a value of one corresponding to the attribute's value. For example, *size* is a feature that has three values *small, medium, large.* This will be converted into three binary features as *size_small, size_medium, and size_large* representing the sparse-coding of {1 0 0}, {0 1 0}, and {0 0 1} respectively.

The sparse-coding representation leads to some loss of information such as features can only have one value at a time. This results in large network architectures. However, discretisation of attributes can reduce learning complexity in neural networks and assists in understanding the dependencies among attributes and target concepts. This type of representation makes rule-extraction techniques easier as it constrains the search space to finite states. Many researchers have compared several techniques for discretisation, and have observed from various experiments that pre-discretisation of numeric attributes often leads to a better solution [Dougherty *et al.*, 1995].

The missing values for an attribute with N possible values can also be handled easily in sparse-coding by considering all the N bits as off (zero), all carrying a mid-value (0.5), or all be $1/N$th corresponding to the missing attribute. The first representation reflects the fact that the value is not known, the second representation uses the mid-value of 0.5 to represent unspecified inputs, and the third representation spreads the value of 1 across all the bits representing the attribute whose value is missing. Shavlik *et al.* [1991] investigated the performance of the three approaches on a number of data sets. The third representation ($1/N$) was found to give the best results as the other two techniques provide too little or too much input activity for the attributes whose value is missing. Therefore, the third representation is used in the experiments (discussed in chapter 6) for data sets that have missing values.

Normalization of data

Many inductive learning algorithms learn and generalize better when data is distributed in a smaller subspace [Weiss and Kulikowski, 1991]. Larger valued inputs (measured in the thousands) have more influence over the smaller valued inputs. As in the case of ANNs which use a sigmoidal function, large input values can saturate neuron's output and thus reduce weight updating and so slow the learning. A solution of this is normalization that can be performed by rational division or subtraction to constrain the input values within the lower and upper bounds. Although simple division or subtractions may work for most cases of normalization, a better understanding is necessary to properly normalize attribute values for the given application domain. The redistribution of data points should be such that it maintains the proportional distance among data points and the distance between two extremes (min_i and max_i). In the GYAN methodology, normalized values of a continuous attribute (without disturbing the Euclidean distance among data points) are calculated by determining the centroid (μ) value and the standard deviation (σ). The centroid and standard deviation of the k^{th} attribute, that has n elements, are

defined as:

$$\mu_k = \frac{1}{n}\sum_{i=1}^{n} v_{ki}$$

$$\sigma_k = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_{ki} - \mu_k)^2}$$

$$v'_{ki} = \frac{v_{ki} - \mu_k}{\sigma_k}$$

$$v''_{ki} = v'_{ki} - min(v'_{ki})$$

v'_{ki} is the normalized value that maintains the distance between each value of the k^{th} attribute and its centroid, where $i = 1, \ldots, n$. If a normalized value turns out to be negative for an attribute, then the minimum value of the attribute is subtracted from all the normalized values. This is to avoid an adverse affect on the neural network's learning, from negative input values [Fausett, 1994].

5.1.3 Data distribution

After preprocessing and transforming data into an appropriate form, data is usually distributed into non-overlapping subsets to test the generalization capability of learnt classifiers. The set of available examples is partitioned into three disjoint subsets: a training set, to adjust the network's parameters (weights, etc.); a validation set, to evaluate the quality of the network during training and to prevent over-fitting of data (i.e. if the network's predictive accuracy degrades on the validation set then the network's construction ceases); and finally a test set, to evaluate the resulting network at the end of training.

K-fold cross-validation (CV) partitioning is then carried out for the data distribution. The k-fold CV randomly divides the available data into k equal-sized, disjoint partitions [Mitchell, 1997]. Each partition in turn is used as a validation set, another randomly chosen partition is used as a test set and the remaining partitions are used as a training set. The data partitioning process is determined by the amount of data, level of noise, missing values, etc. More specifically a *three-fold cross validation* is used for data sets with over *1000*

instances; a *five-fold cross validation* is used for data sets consisting of *200 to 1000 instances*; and for data sets with fewer than *200 instances*, a *ten-fold cross validation* is carried out.

5.2 Application domains

This section briefly discusses the problem domains that are considered for evaluation of GYAN. All of the data sets, except the Remote Sensing and Queensland Rail, were downloaded from the UCI ML Repository [Murphy, 1995]. A fundamental reason to choose the problem domains from the UCI ML Repository is that the prominent features of most of the data sets and the experimental results performed by other researchers are available for comparison purposes. All the data sets are concerned with the *supervised classification learning task*. In classification learning, the task is to predict *discrete-valued outputs* for the given inputs, in contrast to the prediction of continuous values in regression learning.

5.2.1 Queensland Rail data set

The Queensland Rail (QR) database was obtained from Queensland Rail, a government department of the State of Queensland, Australia. It provides information on the risk assessment of QR level-crossings. The objective is to assist QR personnel to improve safety measures to avoid level-crossing accidents by identifying accident-prone cases. Each instance describes the characteristics of level-crossings in Queensland, which are labeled as *Risky* or *Safe* depending on whether or not an accident has happened at the corresponding crossing. In the original data set, there were 26 attributes, 176 *Risky* cases and 3615 *Safe* cases. The goals are to identify the features, either individually or in groups, that are responsible for the risk of a level crossing, and to interact with the user to identify the future (unseen) cases as *Risky* or *Safe*. As a result, the types of protections required to achieve an acceptable level of risk will be identified.

Data preprocessing

The original QR data base contains two relational data sets: accident history, and level crossings. The level crossing data set contains all the important information about level-crossings, surrounding situations and the vehicle (train) itself. The accident history data set contains information about all the accidents that have happened at level crossings in Queensland, Australia. The accident data was only utilized to categorize the level-crossing instances (based on the matching of the unique *Level crossing ID*) according to the constituent value {accident or not} in one of its fields, *consequence type*. The non-meaningful attributes, such as *Level crossing ID, FMS branch name, LS code, Nearest Station, Km, Road Name, Source, and Comments*, were excluded from the data set as they were only stored for identification purposes and were not expected to contribute towards the final outcome.

The level crossing data contains noise because of the different sources used in collection. Most of the discrepancies in data were caused by differing coding schemes. For example, the *Pedestrian density* attribute had values in {*high, medium, low, yes, no*} and the *School Children* attribute had values in {*low, medium, yes, no*}, where the *'yes'* values should be one of *'high', 'medium'* or *'low'*. Such logically impossible or inapplicable values were replaced by the correct values (more generally by the most frequent value).

A large portion of desirable data is missing (unavailable) and most is impossible to retrieve as they are collected from operational data. There were some attributes that had very low distribution against other attributes in the data set because of missing values, for example the attribute *Pedestrian protection* was specified in only 1.2% of cases. This type of attribute is just omitted from the data set if there is no significant number of patterns containing this attribute falling into the category of *Risky*. A deliberate decision had to be made whether to overlook a missing value or to delete it or to replace it. Firstly, the relative distribution of values in the attributes is determined to handle missing

Table 5.1: The attribute domain for the level-crossing data set

Attribute	Type	Attribute domain
Type	Categorical	public, occupation, pedestrian, qr, removed, stock, tramway
Protection	Categorical	boomgates, closed, flashing, fenced, signs, others, none
Traffic lights	Categorical	yes, no
Signs	Categorical	giveway, pedestrian, stop, triangle, x-crossbuck, xg, xs, xt, others
Tracks	Categorical	one, two, threeORmore
Surface	Categorical	bitumen, concrete, dirt, gravel, rubber,tarcol
Sleeper	Categorical	concrete, steel, timber
Train speed	Continuous	0 to 160
Road visibility	Categorical	fair, good, poor
Rail Visibility	Categorical	fair, good, poor
Orientation	Categorical	east-west, north-south
Intersection	Categorical	angled, s-bend, t-intersect, rightangled
Pedestrian Density	Categorical	high, low, medium, nil
School Children	Categorical	yes, no
Lighting	Categorical	yes, no
Approach Signs	Categorical	yes, no
Target attribute	Categorical	Risky, Safe

values. If there was a large percentage of missing values for an attribute (more than 30%), the lack of information was treated as a valuable indication, and was considered as a special value to be included additionally in the attribute domain. If not many (less than 15%) values of an attribute were missing, the missing values were simply disregarded, and during data-transformation these values were given consideration. For example, all the bits in the sparse-coded representation were set to $1/N$ for the missing input value of an attribute with N possible values, rather than applying normal coding as explained in section 3.1. In the cases where more than 11 values were missing out of the total 16 values, the pattern was omitted if it was representing the *Safe* case.

As a result, several changes were made to the level-crossing data and finally the duplicated instances were removed. The resultant data set consists of 16 attributes, and has a total of 1734 instances, 122 of them indicating the target

Risky class and the remaining 1612 representing the *Safe* cases. The attribute domain for the data set is listed in Table 5.1.

5.2.2 Remote Sensing data set

The Remote sensing image data [Hammadi and Korczak, 1995; Wong, 1997] is a high resolution scene of Strasbourg in France obtained from the earth observation satellite SPOT XS. The goal of the Remote sensing problem domain is to recognize the existence of *urban areas* (such as roads, city etc.), *cultivated areas* (such as fields etc.), *natural areas* (such as forest etc.), *water areas* (such as lakes etc.) and the recognition of structured objects from radiometric pictures. The data consists of 64K instances corresponding to 256 * 256 pixels, 1 byte per pixel. Each instance holds three inputs in the range of 0 to 255 corresponding to the pixel gray level value of one of the three radiometric pictures, and one output category (*urban, cultivated, natural, water*) according to the classification by human experts.

The gray level values are quantized into five ranges: (0,..,50), (51,..,80), (81,..,110), (111,..,160), and (161,..,255) to reduce the dimensionality of the problem. Further the attributes have been categorized in sparse-coding as:

Attribute	**Values**
Red	very_low, low, medium, high, very_high
Green	very_low, low, medium, high, very_high
Blue	very_low, low, medium, high, very_high

There were many duplicate instances due to compression from 256 continuous values to the 5 discrete values. As a result, 85 instances are left for training from the initial 8k instances, and 76 for testing and validation from the initial 56k instances. Both the training and testing data sets were processed separately, so there is the possibility of overlapping patterns.

5.2.3 Monks data set

The Monks domain comprises three distinct problems, Monk1 is a selection and equity problem, Monk2 is a parity problem and Monk3 is an exclusion problem. The Monks domain is an artificially created disjoint data set that consists of the following attributes and their values:

Attribute	**Values**
Head_shape	round, square, octagonal
Body_shape	round, square, octagonal
Is_smiling	yes, no
Is_holding	sword, balloon, flag
Jacket_color	red, yellow, green, blue
Has_tie	yes, no

The Monk problems are fairly simple (medium) learning tasks with a relatively medium size finite hypothesis space. The instance space for a Monk problem contains exactly = 3*3*2*3*4*2 = 432 distinct instances. For each of the Monks problems, the data set has already been selectively partitioned into a training and a test set [Thrun *et al.*, 1991]. For the Monk3 problem domain, 5% class noise (the output of six instances is corrupted) has been deliberately added to the training set.

Problem Domain	**Train set**	**Test set**
Monk1	124	308
Monk2	169	263
Monk3	122	310

5.2.4 Mushroom data set

The target of the Mushroom problem domain is to identify *poisonous* mushrooms based on the information provided about an individual. The original mushroom data set, consisting of 22 attributes and 1 output class (*edible* or

poisonous), has 8124 instances (4208 *edible*, 3916 *poisonous*). If sparse coding is considered for the input space in ANNs, the number of nodes yielded is 117. To reduce the size of the hypothesis space, a functional dependency test [Geva and Orlowski, 1996] is performed on the data set before presenting it to the neural networks. A functional dependency algorithm identifies a subset of inputs that can successfully determine the output. The application of this algorithm results in a significant reduction in the size of the input space. The subset of attributes important to decide the *edible* mushroom, as determined by the functional dependency algorithm is {*Cap-colour, Odour, Stalk surface above ring, Spore print colour, Population, Habitat*}. There were many duplicate instances due to compression of input dimensionality from 22 attributes to these 6 attributes. After the removal of duplications and conflicts, the total number of instances is reduced from 8124 to 269.

The original Mushroom domain (22 attributes) is also used to train ANNs, and then the hypothesis space is reduced by pruning the trained networks before rule-extraction.

5.2.5 Voting data set

This data set includes votes for each of the 1984 United States House of Representatives congressmen on the 16 key issues identified by CQA (Congressional Quarterly Almanac, 98th Congress). The task is to identify the vote as *Democrat* or *Republican* based on the information provided. The total number of instances is 435 (267 *Democrats*, 168 *Republican*), and the total number of attributes is 17 including the target class (all boolean values). The independent attributes have values - 'yes' for *voted for* and 'no' for *voted against.* A total of 24% of the values are missing in the instance space, indicated by '?'. There are some missing values for each of the attributes. This lack of information was treated as a third category 'unknown disposition' for *did not vote or voted with conflicts.*

5.2.6 Moral Reasoner data set

The Moral Reasoner data set comes from a rule-based model that qualitatively simulates moral reasoning. The model was intended to simulate how an ordinary child, about age five, reasons about harm-doing. The 202 instances (102 positive, 100 negative) are given as flat (not hierarchical) Horn-clause theory, *i.e.* specific rules using ground predicates. Each rule contains one consequent predicate and 23 antecedent predicates. The top-level predicate to predict is *guilty*, and this is represented as a unary relation. Each antecedent predicate is represented as a binary relation, where the first argument is a person or instance, and the second argument is instantiated by its value. During preprocessing, the relational data base is transformed into attribute-value language. Each relation in the antecedents level is considered as an attribute, and all the possible values for which the second argument of this relation holds are considered as this attribute's domain. The sign of the consequent predicate in a rule determines the instance to be either positive or negative. The total number of instances is 202 (102 *guilty* and 100 *not-guilty*), and the total number of attributes is 24 including the target class (19 boolean values and the others categorical).

5.2.7 Cleveland Heart Disease data set

The goal of the Cleveland heart disease database is to find the presence of heart disease in patients based on thirteen attributes such as cholesterol reading, chest pain etc. The original data set has 76 raw attributes, but only 14 of them have actually been used in past experiments [Murphy, 1995]. The data set, consisting of 13 attributes and 1 output class (patient *healthy* or *sick*), has 303 (164 *healthy* and 139 *sick*) instances. Some of the attributes are in the continuous-valued domain, and have been discretised or normalized to satisfy the requirements of machine learning classifiers. The accumulated frequency graphs drawn for the continuous attributes to determine intervals are shown in Figure 5.1.

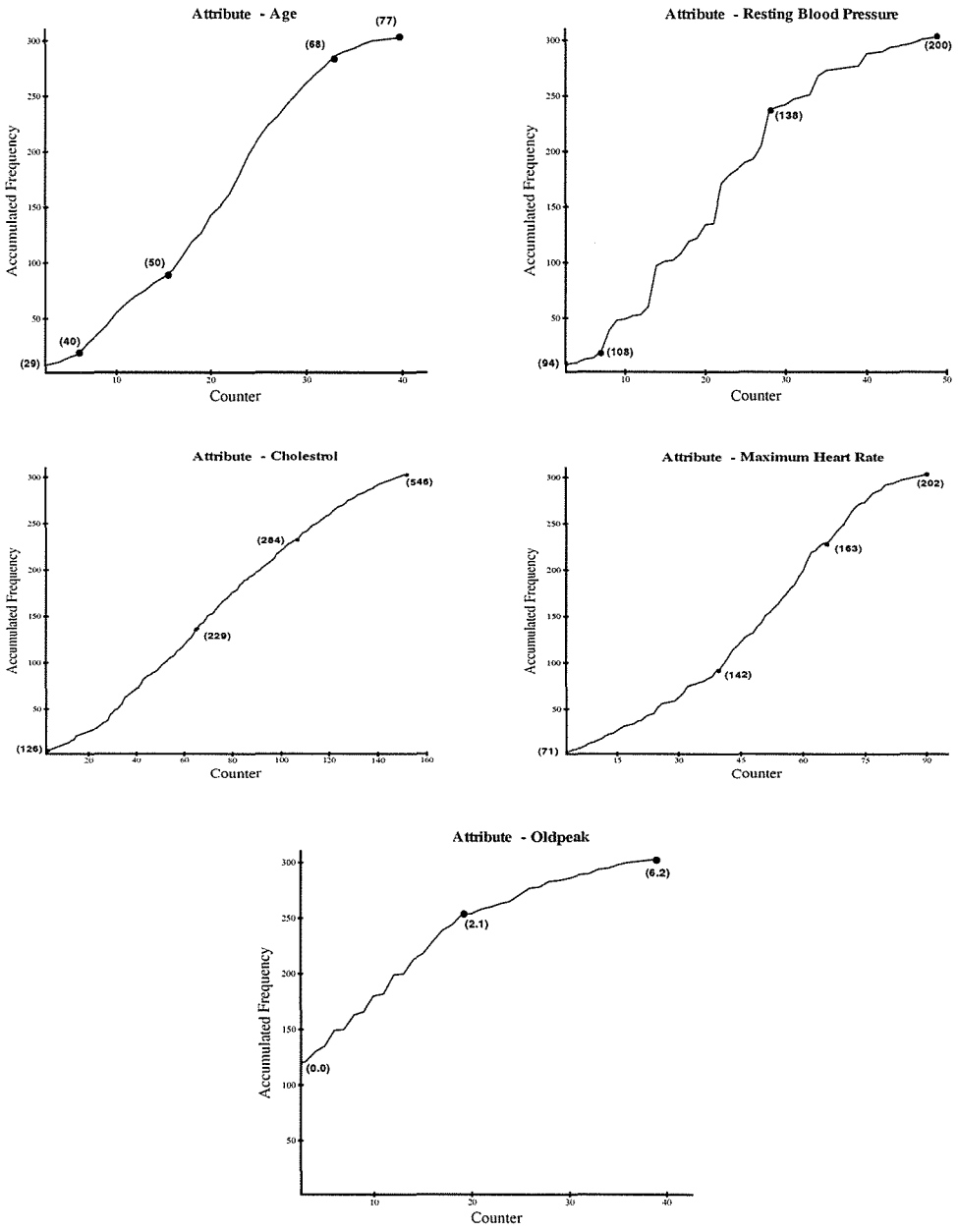


Figure 5.1: Statistics of the continuous attributes in the Cleveland Heart Disease Domain. Each continuous attribute is first sorted in ascending order. The individual occurrence of each instance in the instance space, and then the accumulated frequency, are counted. A graph of accumulated frequency is drawn for each attribute. The relevant interval boundaries according to actual value of the attributes are shown in graphs.

5.2.8 Breast Cancer data set

The target of the Breast Cancer database is to distinguish between the *benign* and *malignant* type of cancer according to nine attributes such as *Clump thickness, Cell size, Cell shape, Adhesion, Bare nuclei, Nucleoli, etc.* The data set has 699 instances from which 16 instances were removed due to missing information about the *Single epithelial cell size* attribute. All the eliminated 16 patterns were instances of *benign* type, that already has a major distribution in instance space. The resulting data base contains 683 attributes, 444 of them are of *benign* type and the remaining 239 are of *malignant* type of cancer.

5.3 Evaluation criteria for inductively generated results

This section describes the criteria for evaluating the GYAN methodology. The task of representing an ANN as predicate rules can be explained as:

Given:

- A set of examples.
- ANNs to learn the target function.
- A rule-extractor to generate predicate rules from ANNs.

Find:

- Concept descriptions (a set of attributes defining a predicate).
- An adapted knowledge base such that all the examples are covered.

Each phase in GYAN is discussed in Chapter 6, with special reference to the following points:

- Analysis of results of the learned model.
- Application of results of the learned model to unseen instances in the data set.

5.3.1 Neural learning algorithms

K-fold cross-validation is used in the experiments to train ANNs for a problem domain. A cross-validation experiment consists of the following steps:

- Randomly divide the available data into k equal sized, disjoint partitions.

- If a separate test set is already given, then for each partition dynamically build a feedforward network according to the parameter file using all the patterns outside the partition, and use the patterns in the partition for validation of the network. The patterns in the test set are then used for evaluation of the network. If there is no separate test set then a random partition is used as a validation set, another random disjoint partition is used as a test set, and the remaining patterns are used as a training set. All initial learning rules and initial network architecture (Cascade and BpTower) are same apart from the random initial weight set for each network. Each of the networks starts with an input layer and an output layer, and dynamically inserts the hidden nodes in the network during learning.

- Choose the network with the highest accuracy and lowest errors for further rule-extraction processing, and report the network's RMSE and classification accuracy for the training set (Train), validation set (Valid) and the test set (Test). Also sum the RMSE over all nets and divide by the total number of built networks to compute the average RMSE. Report this average RMSE and classification accuracy with the standard deviation. The standard deviation for the accuracy is illustrated in the form of an error bar.

The number of epochs taken to reach cessation of training is deemed irrelevant as the incremental training algorithms are used to construct networks. The rule extraction process is continued for the selected network that produces the lowest classification error on training, validation and test sets.

5.3.2 Rule-extraction techniques

The *'quality of extracted rules'* is one of the evaluation criteria for the proposed system GYAN to measure how well the task of extraction has been performed. According to the Andrews, Diederich and Tickle [1995] taxonomy of rule-extraction, criteria for evaluating a rule set include: *accuracy, fidelity, consistency, and comprehensibility.* The *accuracy* of a rule extraction method is a measure of the generalization ability of the extracted rule set on unseen data (how well the data is classified by the extracted rule set). The accuracy of a rule-extraction technique can be expressed as:

$$Accuracy = 1 - \frac{Number\ of\ mismatches\ in\ rules}{Total\ number\ of\ instances\ to\ be\ classified}$$

The *fidelity* of a rule extraction method is a measure of the agreement between the network and the extracted rule set (how well the extracted rule set mimics the behavior of network). The Fidelity of a rule-extraction technique can be expressed as:

$$Fidelity = 1 - \frac{No.\ of\ mismatches\ in\ rules - No.\ of\ mismatches\ in\ network}{Total\ number\ of\ instances to be classified}$$

The *consistency* of a rule extraction method is a measure of the agreement (as 'yes'/'no') among various extracted rule sets from neural networks under differing training sessions (how well the unseen data is classified by extracted rule sets under different training circumstances). The *comprehensibility* of an extracted rule set is a measure of the number of rules, antecedent predicates and arguments in the consequent predicates. This criterion is extended for automated knowledge bases by measuring the number of facts, instances in the type-hierarchy, the depth of queries and the number of steps taken to solve a query. The rule quality criteria are extended by an additional dimension that measures the *improved generalization.* The *improved generalization* of a rule extraction method is a measure (as 'yes'/'no') of better generalization performance of the extracted rule set over the neural network from which the rule set was extracted. If the extracted rule set performs better than the un-

derlying network then the *improved generalization* is considered to be 'yes' otherwise 'no'. Some of the experimental studies [Towell and Shavlik, 1993; Alexander, 1994; Nayak *et al.*, 1997] have noticed that sometimes extracted rules exhibit a better generalization performance than the trained neural network from which they are extracted because the extracted rules help to suppress noise and provide constraint that bounds generalization.

5.4 Chapter summary

The preceding discussion on data representation has shown the importance of preprocessing of data before applying it to a inductive learner. It also shows the methods of data transformation for an equivalent representation required by the component programs in GYAN. All the application domains used in the evaluation of GYAN are introduced, along with the preprocessing that the individual data sets required in order to be amenable to component programs. The last section describes the dimensions in which this methodology should be evaluated. The evaluation criteria will allow comparison of each component in the methodology, and will help to decide the best technique (decompositional or pedagogical) suitable for certain tasks. In summary, this chapter has laid the foundation of the empirical analysis of GYAN carried out in the following chapter.

Chapter 6

Experimental evaluation

Chapter 4 has presented a detailed description of the GYAN methodology, which provides a symbolic explanation of the knowledge embedded in trained ANNs. The preceding chapter on data analysis and representation has set the foundation of the experimental analysis of GYAN. This chapter provides the empirical evaluation of GYAN. The main focus of this chapter is:

- to empirically investigate the idea involved in GYAN; that is, the knowledge embedded in trained ANNs can be represented as restricted first order logic rules with high accuracy and without loss of fidelity.
- to validate the idea in GYAN that the knowledge embedded in ANNs can be processed to be used in a knowledge base reasoner to allow user explanation by evaluating the quality of knowledge bases generated by GYAN in terms of comprehensibility, accuracy and fidelity.
- to provide a basis for comparison of results obtained by different component programs in GYAN in any phase *i.e.* bottom-up (BpTower) vs top-down (Cascade) neural learning approaches, and decompositional (*LAP*) vs pedagogical (*RuleVI*) rule extraction techniques.
- to provide experimental validation that, projection to a space of a lower number of input dimensions by pruning allows translation of the trained ANN into a compact rule set that sometimes seems impossible to obtain from the original high input dimensionality.

- to evaluate the efficiency of GYAN against other techniques such as FOIL [Quinlan, 1990].

The experiments presented here illustrate the application of GYAN to various problem domains discussed in the preceding chapter. The Queensland Rail and Remote Sensing problem domains are selected to show the applicability of GYAN to operational and real-life problems. The Monks, Mushroom and Voting problem domains are selected to compare the efficiency of GYAN with the experimental results reported by other researchers. The Moral Reasoner data base is chosen to demonstrate the similarity in the knowledge base automated by GYAN and the given Horn-clause theory. The Cleveland heart disease problem domain is to demonstrate the applicability of GYAN to mixed-valued domain problems (discrete and continuous-valued attributes). The Breast Cancer problem domain is to demonstrate the effectiveness of GYAN to fairly large (in terms of input space) problems. Each phase in GYAN is empirically evaluated separately.

6.1 Gyan applied in phase1

In this section, performance of various ANNs is evaluated against the set of criteria presented in the preceding chapter. One of the goals in developing GYAN is the applicability of predicate (or restricted first-order) rule-extraction to a variety of ANN architectures. This goal is achieved by integrating several neural learning techniques such as cascade correlation (CC), BpTower (BT) and constrained error back propagation (CEBP) within the framework. Data from a problem domain is presented in a sparse coded scheme to the cascade correlation and BpTower architectures to facilitate pruning and rule extraction in the subsequent phases. The CEBPN architecture does not require a special type of encoding scheme to represent data, except for the general ANN requirement that data should not be distributed over large ranges.

The main focus within GYAN Phase1 is to construct ANNs with a small

number of hidden units and a high percentage of correctly classified test patterns for the given problem domain. This involves the selection of certain parameter settings such as: the type of activation functions in hidden and output layers, ε and μ for weight updating in the case of cascade learning; learning rate and momentum in the case of BpTower network training; and sigmoidal edge steepness of basis function ridges in the case of CEBPN learning. There is also the selection of termination parameters such as the number of maximum inserted nodes, minimum RMS error on training patterns and the number of training epochs to use in a learning cycle to decide when the further growing of ANNs should be stopped.

6.1.1 ANN solutions to the problem domains

During BpTower and cascade learning, a few soft constraints are imposed upon the ANNs. One of the constraints is to limit the precision of the modified weights to restrict the search space for a rule extraction technique (chapter 4, section 4.2.1). To empirically find out which level of precision should be used in weight values, a number of experiments were performed on several data sets with different level of precision (namely values are rounded to integer, first, second or third decimal places) for each weight value. In general, the best results were obtained in terms of the size of ANNs and predictive accuracy, when weight values were rounded to second decimal places. Table 6.1 reports the RMSE and the number of hidden nodes required to produce near-optimal weight states with different levels of precision.

Tables 6.2 to 6.12 report the performance of ANNs on the data sets discussed in chapter 4. The column 2 in these tables is the ANN architecture, *i.e.* the number of input, hidden and output nodes. The number of nodes in the input layer is the total number of values of all the attributes in the data set, as inputs to the cascade (CC) and BpTower (BT) networks are sparsely coded. The number of nodes in the input layer for CEBPNs corresponds to the input dimensionality of the given problem. The column 4 in these tables is the

Table 6.1: Performance of cascade correlation networks with different levels of precision in weight vectors. In the case of XOR (2 inputs or 3 inputs), the evaluation set is the same as the training set. The reported classification accuracy is the average accuracy on the training and test sets for all the data sets.

		Precision level 3	Precision level 2	Precision level 1	Precision level 0
XOR2	RMSE	0.0511	0.0688	0.2695	0.3155
	Hidden units	1	1	2	4
	Accuracy (%)	100	100	100	75
XOR3	RMSE	0.0311	0.0341	0.1582	0.308
	Hidden units	1	1	2	4
	Accuracy (%)	100	100	100	75
Monk1	RMSE (test)	0.018	0.019	0.0997	0.3314
	Hidden units	1	1	5	5
	Accuracy (%)	100	100	99	85
Monk2	RMSE (test)	0.059	0.0615	0.1238	0.1432
	Hidden units	1	1	5	5
	Accuracy (%)	99.76	99.76	98.5	94
Monk3	RMSE (test)	0.210	0.2379	0.2549	0.3178
	Hidden units	1	1	6	6
	Accuracy (%)	95.14	95.14	92.39	89.8

classification (or class'n) mismatch, that is the ratio of incorrectly classified instances to the total number of instances (training or testing). The pruning process removes nodes and links from the ANN that are not contributing towards the ANN's prediction, resulting in a reduced size network (PCC - pruned cascade network, PBT - pruned BpTower network).

Monks problem domain: For each of the Monks problems, the domain has already been selectively partitioned into the training and test sets by the

Table 6.2: Performance of ANNs on the Monk1 data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	17:1:1	0.01599	0.01908	0/124	0/308
PCC	8:1:1	0.03348	0.03341	0/18	0/18
BT	17:1:1	0.02733	0.06433	0/124	0/308
PBT	10:1:1	0.0234	0.0535	0/36	0/36
CEBPN	6:6:1	0.02142	0.04811	0/124	0/308

Table 6.3: Performance of ANNs on the Monk2 data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	17:1:1	0.0059	0.0615	0/169	1/263
PCC	12:1:1	0.0061	0.0616	0/52	1/60
BT	17:1:1	0.0246	0.0667	0/169	1/263
PBT	12:1:1	0.0547	0.0543	0/52	1/60
CEBPN	6:6:1	0.01188	0.2585	0/169	66/263

Table 6.4: Performance of ANNs on the Monk3 data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	17:1:1	0.1022	0.2379	1/122	20/310
PCC	9:0:1	0.2053	0.0228	1/24	0/25
BT	17:2:1	0.1283	0.1496	3/122	6/310
PBT	12:0:1	0.2058	0.0735	3/62	0/74
CEBPN	6:4:1	0.0573	0.0290	7/122	9/310

Table 6.5: Performance of ANNs recognizing a water area in the Remote Sensing problem domain

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	15:0:1	0.02018	0.02099	0/85	0/76
PCC	13:0:1	0.091	0.01	0/75	0/68
BT	15:0:1	0.0247	0.0260	0/85	0/75
PBT	13:0:1	0.0437	0.0215	0/75	0/68
CEBPN	3:2:1	0.0468	0.00001	0/85	0/75

Table 6.6: Performance of ANNs recognizing forest area in the Remote Sensing problem domain. There is no reduction in the size of ANNs after pruning.

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	15:0:1	0.0392	0.0298	0/85	0/76
PCC	15:0:1	0.0411	0.0301	0/85	0/76
BT	15:0:1	0.0692	0.0317	0/85	0/76
PBT	15:0:1	0.0737	0.0516	0/85	0/76
CEBPN	3:3:1	0.0832	0.0011	0/85	0/76

originators of the domain. The training instances are uniformly distributed for each of the Monks data sets. Five ANNs were trained on the same training set with randomly initialized weights for each of the Monks problems. Tables 6.2, 6.3 and 6.4 show the performance of the best ANNs for each of the data sets. The classification accuracy of the trained ANNs is quite high for the Monk1 and Monk2 data sets. The ANNs trained for the Monk2 problem domain have learned the Monk2 concept except when all the six attributes have their first values. The reason is that during training the ANNs have not seen the supporting instance that contains the first value for all the 6 attributes. As a result, the BpTower and the cascade networks are not able to classify robots as 'not monk' that have the first value for all six attributes. The classification accuracy is relatively lower for the Monk3 data set due to the deliberate 5% noise added in the training patterns (the value of the target attribute is corrupted for the six instances).

Remote Sensing problem domain: The Remote Sensing problem domain has been separately partitioned into the training and test sets before the quantization of data. There are very few instances in the quantized problem domain that represent the target class, water. However, due to disjoint partitioning of training instances across the problem space (a linearly separable problem), the trained ANNs are capable of correctly approximating the target concepts for this problem. Ten ANNs were trained on the same training set with randomly initialized weights for the recognition of water and forest areas in the Remote Sensing problem. Tables 6.5 and 6.6 :how the performance of the best ANNs, chosen for their highest accuracy and lowest RMS error. Based on the ANN solutions, it can be said that the Remote Sensing problem domain has been transformed into a linearly separable classification problem after compressing (quantizing) the input space from 256 continuous values to 5 discrete values.

Mushroom problem domain: 3-fold cross-validation tests were per-

Table 6.7: Performance of ANNs on the Mushroom data sets - the original data set with 22 attributes and the data set with selective 6 attributes. The classification mismatch (column 4) for Testing is given as the ratio of instances that are incorrectly classified over the total number of instances present in validation and testing sets.

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC (22)	117:0:1	0.0010	0.0010	0/2708	0/5416
PCC (11)	29:0:1	0.0148	0.0107	0/42	0/42
CC (6)	45:0:1	0.0092	0.0054	0/163	0/106
PCC (6)	24:0:1	0.0243	0.0042	0/81	0/28
BT (22)	117:0:1	0.0049	0.0054	0/2708	0/5416
PBT (10)	25:0:1	0.05405	0.01865	0/54	0/54
BT (6)	45:0:1	0.0725	0.0198	0/163	0/106
PBT (6)	19:0:1	0.05032	0.00999	0/48	0/48
CEBPN (22)	22:3:1	0.0677	0.07425	14/2708	28/5416

formed to train the BpTower, Cascade and CEBP networks with randomly generated training patterns and randomly initialized weights on the original Mushroom problem domain, which contains 23 attributes (including the target attribute). The results reported in Table 6.7 for the CEBPN were obtained at the 150th epoch presentation, and still contain some prediction error. There was little gain in continuing to train for much longer than this. By adding a few more bumps (local hidden units) the classification mismatch error on the training and test sets goes down, but the rule-comprehensibility becomes poorer in a later phase of GYAN. The optimum CEBPN solution has 3 local hidden units, corresponding to one positive (edible) rule and two negative (poisonous) rules. The Mushroom data set contains a large number of values in attributes, resulting in a big input space when sparse-coded. To obtain a suitable subset of this data set, the functional dependency algorithm [Geva and Orlowski, 1996] was applied to one of the training sets consisting of 2708 instances. A total of 119 subsets were obtained, each consisting of 4 to 8 dependent attributes. From these, a subset consisting of 6 attributes was selected for the application of GYAN. Table 6.7 shows the reduced ANN size, RMS error and the accuracy of the best cascade and BpTower networks trained on the compressed data set using a 5-fold cross-validation scheme.

Table 6.8: Performance of ANNs on the Voting data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	48:1:1	0.057	0.148	0/261	6/174
PCC	24:1:1	0.098	0.1977	1/158	9/79
BT	48:1:1	0.0099	0.1551	0/261	4/174
PBT	28:1:1	0.0137	0.1029	0/169	10/84
CEBPN	16:11:1	0.0505	0.0264	0/261	7/174

Table 6.9: Performance of ANNs on the Moral Reasoner data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	48:1:1	0.0253	0.0386	0/162	0/40
PCC	26:0:1	0.0454	0.0123	0/151	0/40
BT	48:1:1	0.0468	0.0532	0/162	0/40
PBT	30:0:1	0.0607	0.1010	0/160	0/40
CEBPN	23:9:1	0.0001	0.2236	0/162	2/40

Voting problem domain: The ANNs were trained separately using each of the BpTower, Cascade correlation and CEBP algorithms for the Voting problem domain (Table 6.8), using a 5-fold cross-validation (CV) scheme. The instances are uniformly distributed across the problem space. In the case of all three neural learning algorithms, the ANN solutions suffer from the problem of overfitting if the training continues, *i.e.* the classification error on the validation set starts to increase while still reducing the remaining error on the test patterns. The best CEBPN solution for the voting data set is with 11 hidden units at the 700th iteration.

Moral Reasoner problem domain: The Moral Reasoner problem domain (Table 6.9) utilizes a 10-fold CV scheme to train the ANNs. All the ANNs using cascade and BpTower learning algorithms show consistent behavior and learn quite easily, whereas the CEBPN solutions are not able to correctly classify the unseen instances. As shown in Table 6.9, the best CEBPN solution still has high remaining RMS error, and classifies two (2) unseen instances incorrectly. During CEBPN learning, the network inserts the 9th hidden node

Table 6.10: Performance of ANNs on the Cleveland heart disease data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	38:2:1	0.0953	0.3392	1/216	15/87
PCC	34:2:1	0.0851	0.221	1/206	10/77
BT	38:2:1	0.1313	0.3832	1/216	13/87
PBT	34:2:1	0.0991	0.2765	4/206	7/77
CEBPN	13:7:1	0.270	0.347	16/216	12/87

at the 300th epoch. Training is continued to learn the shape of ridges (or sigmoids) till the 450th iteration. The next ridge inserted in the ANN after that epoch does not improve the performance.

Cleveland heart disease problem domain: Table 6.10 shows the performance of trained ANNs, the result of 5-fold CV tests for the Cleveland heart disease domain. The cascade and BpTower network solutions for this domain produce optimum solutions if the number of inserted hidden nodes is two. With the insertion of more nodes, the classification error on the training and test sets reduces but the error on the validation set increases. Similarly if training is continued for the CEBPN solutions, the network's generalization accuracy degrades or remains unchanged.

Breast Cancer problem domain: The Breast Cancer problem domain (Table 6.11) utilizes a 5-fold CV scheme to produce the ANN solutions. There is significant variation in the size of the generated cascade network solutions; that is, the number of hidden units built into the nets varies in the range of 0 to 4. The goal is to develop a small size feedforward ANN with sigmoidal nodes that properly classifies the training and unseen examples. One of the ANN solutions (Table 6.11) does not insert any hidden nodes and performed better than others. Those which keep trying to classify the remaining patterns correctly by inserting more nodes, result in larger ANNs with poorer generalization (because of overfitting). In contrast, the BpTower network solutions show consistency in terms of number of hidden nodes and classification accu-

Table 6.11: Performance of ANNs on the Breast Cancer data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	90:0:1	0.0054	0.1887	0/411	7/272
PCC	42:0:1	0.0076	0.0951	0/205	3/134
BT	90:0:1	0.0099	0.1107	0/411	9/272
PBT	42:0:1	0.0134	0.1047	0/223	4/150
CEBPN	9:9:1	0.085	0.1917	0/411	10/272

Table 6.12: Performance of ANNs on the QR data set

	Network Architecture	RMS Error		Class'n Mismatch	
		Training	Testing	Training	Testing
CC	63:0:1	0.2364	0.2174	102/1561	14/173
PCC	31:0:1	0.2538	0.2780	43/490	4/95
BT	63:0:1	0.2743	0.2097	104/1561	11/173
PBT	34:0:1	0.2776	0.2564	44/500	3/100
CEBPN	16:5:1	0.2477	0.2633	107/1561	12/173

racy at approximating the target concept. The CEBPN solution reported in Table 6.11 took 550 iterations to sequentially add nine (9) hidden nodes to accurately partition the data set. Training beyond this point generates the ANN which generalizes poorly.

Queensland Rail problem domain: There is considerable difference between the distribution of *Risky* and *Safe* cases in the QR dataset, 93% of the instances belong to the *Safe* class and only 7% of instances belong to the *Risky* class. Also the QR data set is an example of a non-separable problem (non-disjoint distribution of target classes). This poses a problem for ANNs (and for some other machine learning tools as well) to distinguish between the two classes. An internal study was carried out at QUT to handle this problem by (1) copying the instances that represent the *Risky* class several times to create an illusion of equal distribution of classes, (2) conducting leave-one-out cross-validation experiments to make ANNs see most of the *Risky* instances during training, and (3) initializing ANNs with a domain theory. No significant improvement in results in terms of correctly classifying *Risky* cases was observed. Because of the uneven distribution of the instances, 10-fold

cross-validation tests were performed and most of the *Risky* cases (95% out of the total 7% cases) were included in the training set every time. Table 6.12 shows that the remaining RMS error is quite high for all the ANNs, and also some of the instances are incorrectly classified. Analysis of the results shows that most of the misclassified patterns belong to the *Risky* class. Prolonged training (increased iterations, and the insertion of more hidden nodes) does not help to reduce the residual error or to increase the number of correctly classified patterns.

6.1.2 Phase1: Summary and conclusion

Figure 6.1 to 6.3 demonstrate the performance of ANN solutions, produced as the results of n-fold CV experiments, in order to show the variation in behavior of ANNs trained using the cascade correlation algorithm. Figure 6.1 shows the number of epochs taken by cascade learning to achieve the overall ANNs for different data sets. Figure 6.2 shows the classification accuracy obtained by each ANN after training. The root mean square error of each ANN is given in Figure 6.3. Tables 6.2 to 6.12 report the performance of the best ANNs only, as these were the only ANNs that were pruned and from which rules were extracted.

The relative overall performance of each neural learning algorithm is determined by separately measuring the classification accuracy achieved by the ANN solutions on an unseen pattern set (generalization) for each data set, and then calculating the average predictive accuracy across all data sets, for each type of learning algorithm. The BpTower neural learning algorithm gives the *better performance* on these data sets (yielding an overall average of 97.31%) in terms of *generalization accuracy* compared to the cascade correlation learning algorithm (96.65%) and the CEBP neural learning algorithm (93.84%).

The following conclusions can be drawn from these experiments:

- The cascade or tower type of neural learning algorithms result in ANNs

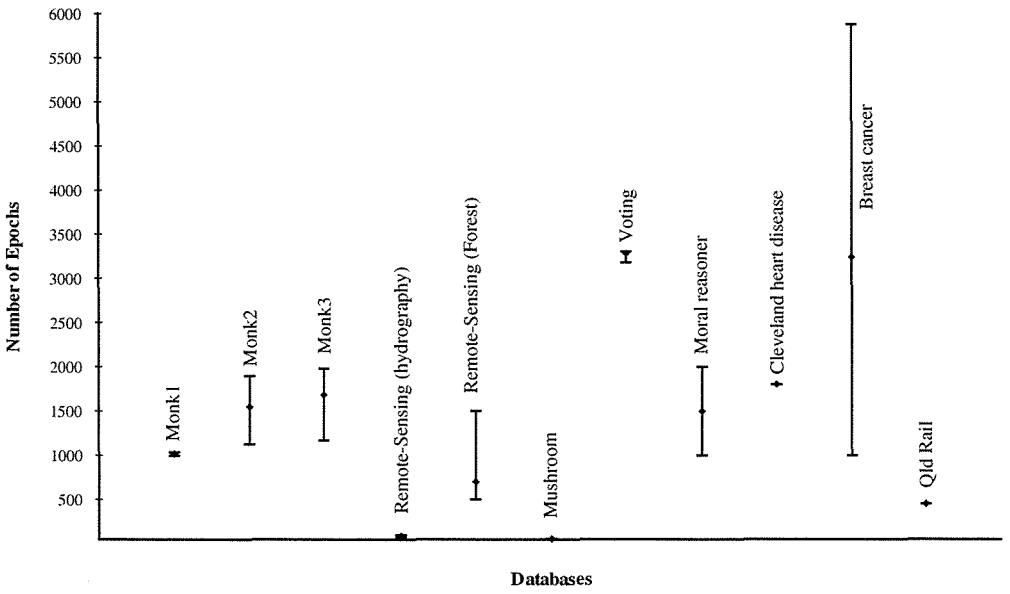


Figure 6.1: Number of epochs used for the ANNs to learn the problem domains

in which the hidden node acts as the dominant feature detector for the problem domain. The CEBP type of learning also results in ANNs in which each hidden node represents the dominant features of the data set.

- The cascade correlation and BpTower algorithms employ different modes of inserting hidden nodes in the ANN during training. As a result, there is a significant difference in the concepts learned by the hidden nodes in each algorithm. The cascade algorithm's hidden nodes try to learn negative concepts, because the output node in the ANN exists and has already learned partial concepts. The BpTower algorithm's hidden nodes try to learn positive concepts, because the hidden node is inserted in the ANN as the output node.

  For example, when the Monk1 problem is induced by both training algorithms, the resulting ANNs consist of a single hidden node with an input layer and a single output node (Figure 4.13). The *jacket-color* attribute with *red* value is the dominating feature in this domain. The hidden node in both the ANNs tried to learn this concept but in differ-

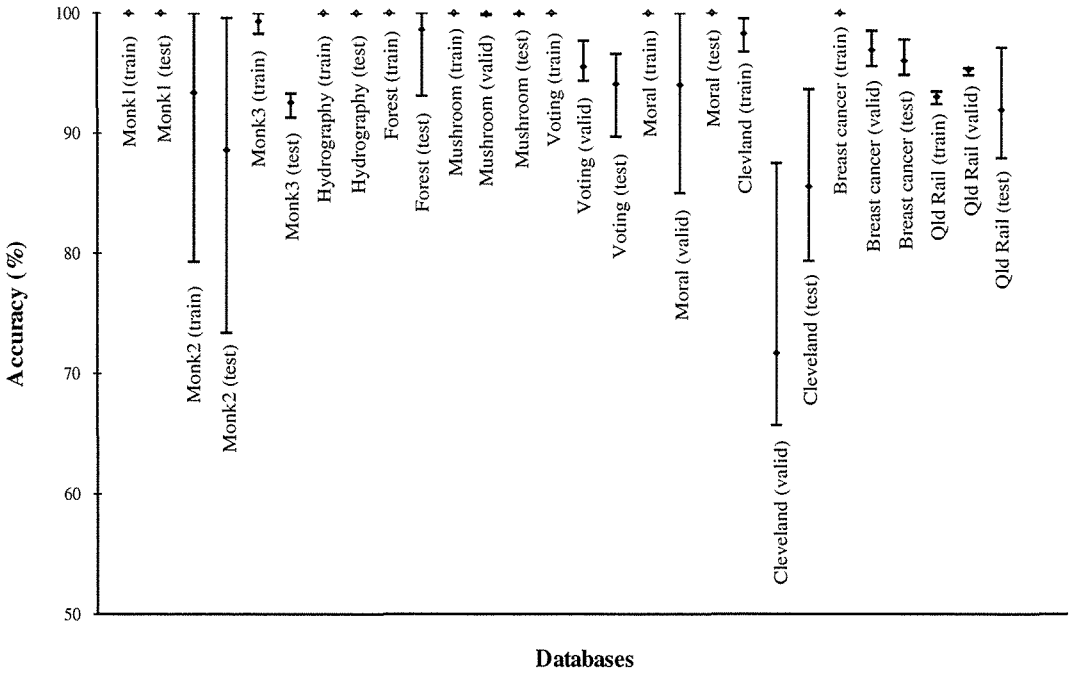


Figure 6.2: Classification accuracy of the ANNs obtained for different problem domains

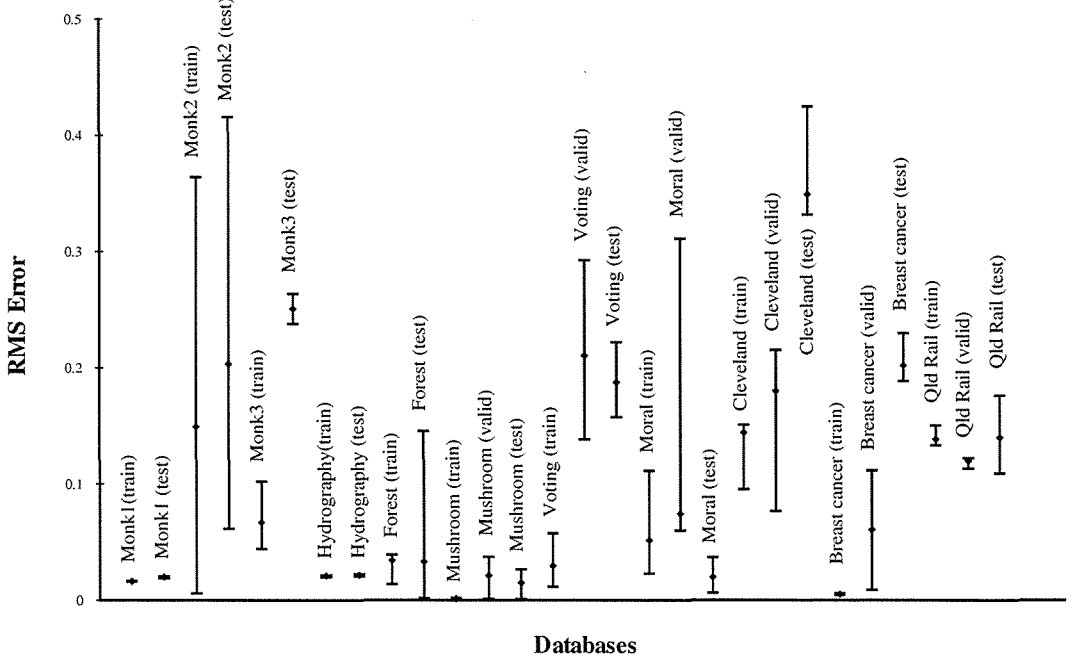


Figure 6.3: RMS error after learning the problem domains

ent ways. The hidden node in the cascade ANN produces a *low output* whenever the *jacket-color* attribute has the value *red*, and contributes a *negative weight of large magnitude to the output node.* On the other hand, the hidden node in the BpTower ANN produces a *high output* whenever the *jacket-color* attribute has the *red* value, and contributes a *large-magnitude positive weight to the output node.*

- The presence of interconnections between hidden nodes in the trained ANNs produced by both cascade and BpTower learning complicates the problem of understanding the trained ANN.

- The cascade correlation network tends to recruit a smaller number of hidden nodes than the BpTower network for a given problem. One of the reasons is that BpTower, like all tower algorithms, approximates the sigmoidal or linear activation function by a Heaviside activation function after the hidden node has been inserted. The pre-existing hidden node then sends approximate output instead of actual output. Consequently, the BpTower algorithm takes longer (more epochs) to train. Also these two ANNs tend to require lesser number of hidden units to approximate a target function than a multi-layer perceptron due to lateral connections. For example, the Monk1, Monk2 and XOR problems are solved by these ANNs (cascade and BpTower) requiring only 1 hidden node, but the backpropagation multilayer ANN requires minimum 2 hidden units.

- The BpTower networks under differing training sessions (randomly initialized weights and patterns) for a problem domain show more consistent behavior than the cascade networks and CEBPN.

- The cascade algorithm tends to produce ANNs in which some weights are distinctly larger in magnitude than other weights in the same nodes. This is encouraged by adding a penalty term during training that decays smaller weights faster than larger weights. This behavior becomes more obvious when weight values are rounded off second decimal place. Due

to these two constraints employed in cascade and BpTower learning, very small weights are already set to zero.

- The CEBPN is constructed by approximating the regions that fit the training data with hyper-ellipses. A solution is not acceptable which makes generalizations about regions of input space for which there is no supporting evidence in the training pattern set. This requires a careful partitioning of instances that are used for training from the total instances.

  The generating solution also very much depends on the edge steepness of the sigmoids (set by the user) that define the local response units. If the steepness is high (4 to 7), the approximation of regions is almost by hyper-rectangles resulting in poor generalization. If the steepness is high (1 to 3), networks can learn and generalize well. But it makes the rule extraction phase (fidelity) poor. The networks that are approximated with high steepness of the sigmoids help to extract high fidelity rules in the subsequent phases.

- In incremental algorithms of neural learning, the selection of termination criteria is important. This becomes more significant when there is noise present in the data or when there is overlapping of instances in the problem space. Beyond a certain point (iteration epochs, number of hidden units, etc.) there is little gain in extended training. If training of the ANN is not stopped at this point, the performance of the ANN either degrades or does not improve.

6.2 Gyan applied in phase2

The main focus within GYAN Phase2 is to obtain a subset of the attributes (part of the input space in an ANN) that can represent the underlying ANN accurately and comprehensively. Once a cascade or BpTower network with minimum training or validation RMS error and maximum classification accuracy has been selected for the problem domain, the pruning algorithm can

be applied to the ANN solution to eliminate irrelevant nodes and links that are not contributing towards the ANN's predictions. Consequently, the attributes/values corresponding to the eliminated nodes are removed from the data set, resulting in a compressed data set that is reduced in the number of input dimensions and in the number of patterns after removal of duplicate instances. The training, validation and test sets are processed independently for the given problem domain, and may contain duplicated instances.

The functional dependency algorithm (FD) [Geva and Orlowski, 1996] that selects a subset of attributes as a *pre-cursor* to the ANN training, is applied to each data set to compare the effect of removing attributes *after* the ANN's training and *before* the ANN's training. The FD algorithm is based on exhaustive identification of functional dependencies among attributes. The algorithm identifies all the existing functional dependencies in the data set rather than just one.

6.2.1 Removal of redundant nodes from ANN solutions

GYAN phase2 includes the selection of a *distance metric* parameter which clusters the weights in trained ANNs to facilitate the pruning of irrelevant links and redundant nodes. Table 6.13 reports the distance metric, the number of clusters that are generated initially with this distance, and the number of clusters (or the remaining links in the ANN, given in brackets) after the pruning procedure is applied to the cascade correlation and BpTower networks. The number of clusters is comparatively large in the networks with hidden nodes, as the output node in the cascade and BpTower networks receives activation from input nodes as well as hidden nodes.

It can be observed from Table 6.13 that a *small distance* is required to group the weight states in the ANNs into a reasonable number of clusters. The rationale is the restrictions imposed upon ANNs during training (GYAN phase1) that clusters the weights in a number of small intervals. This table

Table 6.13: The distance metric and the number of generated clusters according to weights in the ANNs.

Data set	Cascade-Correlation		BpTower	
	Distance Metric	No of clusters generated	Distance Metric	No of clusters generated
Monk1	2.0	11 (7)	0.2	19 (10)
Monk2	2.0	13 (7)	0.25	10 (6)
Monk3	0.15	19 (11)	0.1	22 (17)
Remote-Sensing				
(water)	1.0	6 (4)	0.3	4 (3)
(forest)	3.1	6 (5)	0.1	9 (10)
Mushroom				
(complete data set)	1.5	15 (9)	0.5	10 (5)
(data set after FD)	1.0	8 (6)	0.3	8 (5)
Voting	0.5	22 (13)	0.1	27 (17)
Moral reasoner-reasoner	0.25	19 (11)	0.1	23 (13)
Cleveland heart disease	0.25	51 (39)	0.1	71 (36)
Breast cancer	1.0	44 (21)	0.1	36 (24)
QR	0.75	9 (6)	0.1	10 (7)

also shows that the distance metric is usually *smaller* in the ANNs produced by the BpTower algorithm than in those produced by the cascade-correlation algorithm. In general, the cascade algorithm tends to produce ANNs with weights that are distinctly large in magnitude as compared to other weights belonging to the same node, which makes the weight states easily distinguishable. The BpTower algorithm does not show this type of behavior.

Monk1 problem domain: After pruning, the pruned network's RMSE error (PCC and PBT) is bit higher than the original network. The reason is that the ANN is only retrained for a few epochs after removal of superfluous nodes and links until it performs the equivalent or higher predictive accuracy on train or test cases, and during pruning the weight values are changed (each cluster's value is replaced by its average value). This behavior is consistent in most of the data sets. The input space remaining after pruning of the cascade ANN solution of the monk1 problem domain consists of *Head_shape* = {*round, square, octagon*}, *Body_shape* = {*round, square, octagon*}, *and Jacket_color* = *red.* The pruned BpTower network solution for the Monk1 problem domain

differs in the values of the attributes *Jacket_color*. The BpTower ANN solution could not eliminate the *yellow, green, blue* of *Jacket_color*, instead these values appear as negated antecedents for the learned concept of the hidden node in the ANN. A number of experiments done by other researchers confirm the selection of these three dominant features in the Monk1 data set [Thrun *et al.*, 1991]. The performance of ANNs after pruning is shown in Table 6.2.

Monk2 problem domain: The monk2 problem domain is one where all the attributes are important to induce the target concept. The FD algorithm reports only *one functional dependency including all the 6 attributes*, which does not help to reduce the hypothesis space. In contrast, GYAN phase2 allows testing of the existence of each value in each attribute by considering the sparse-coded input space. As a result, both the BpTower and cascade network solutions for the Monk2 problem domain eliminate few input nodes. The remaining input subspace contains the first values of all the six attributes, *i.e.* *Head_shape = round, Body_shape = round, Is_smiling = yes, Holding = sword, Jacket_color = red, and Has_tie = yes* for both the ANNs. Table 6.3 shows the performance of ANNs after pruning.

Monk3 problem domain: The Monk3-cascade solution after pruning keeps the attributes, *Body_shape* = {*round, square, octagon*}, *Holding = sword, Jacket_color* = {*red, yellow, green, blue*} in its input space. The BpTower solution does not distinguish the values of the *Holding* attribute and keeps all of them in the input space, and also adds the *has_tie* attribute. Table 6.4 shows the improvement in performance in terms of predictive accuracy after pruning. On the other hand, the FD algorithm recognizes only *one functional dependency* among *Head_shape, Body_shape, Holding and Jacket_color.* The actual target of the Monk3 problem domain is: *(Body_shape* = $\neg$*octagon* $\wedge$ *Jacket_color* = $\neg$*blue)*; or *(Holding = sword* $\wedge$ *Jacket_color = green).*

Remote Sensing problem domain: GYAN phase2 removes the un-

wanted *very_low* and *low* values of the attribute *Blue* from both the cascade and BpTower network solutions for the Remote Sensing problem domain recognizing water (Table 6.5). By contrast, the FD algorithm did *not* elicit any functional dependencies among attributes for water recognition. GYAN phase2 did *not* reduce the input space in the Remote Sensing domain recognizing forest for either of the learning algorithms. Similarly the functional dependency algorithm did *not* recognize any dependencies for forest recognition.

Mushroom problem domain: Two different input subspaces are generated for the Mushroom problem domain by performing the functional dependency test on the original data set. This is done to compare the effect of pruning before and after the ANN's training. The cascade and BpTower network solutions contain 117 input nodes to approximate the original data set. The dimensionality is reduced from *117 to 29 entities* for the cascade network solution by pruning to remove the irrelevant attributes or specific values in attributes. The resulting ANN contains nodes involving the attributes/values: *Odour (with 8 values), Gill-size (broad), Gill-color (buff), Stalk-surface-above-ring (silky), Stalk-surface-below-ring (scaly), Stalk-color-above-ring (orange), Stalk-color-below-ring (orange), Ring-type (flaring, large), Spore-print-colour (black, brown, green), Population (clustered), Habitat (leaves).* The pruned BpTower network contains the very same attributes with a little variation in the values, reducing from *117 to 25 input nodes.*

When the FD algorithm was applied to the original Mushroom data set, a total of *119* subsets were obtained, each containing a number of functionally dependent attributes varying in the range from 4 to 8. The BpTower and cascade network contained 45 input nodes by adopting a sparse-coding scheme for the selected subset of attributes *Cap-color (10 values), Odour (9), Stalk-surface-above-ring (4), Spore-print-color (9), Population (6), Habitat (7).* The pruning algorithm was further applied to reduce the input space after training the ANNs. The pruning algorithm was able to remove 21 superfluous nodes,

keeping only the 24 input nodes in the cascade network. Table 6.7 shows the performance of all the ANNs after pruning. The logical rules obtained by other researchers [Murphy, 1995] show that the attributes/values selected by the pruning algorithm, applied on the ANN solutions of the original Mushroom data set, are the only important features, and *all* are necessary to represent the data set as a complete set of rules. The attributes/values selected for the ANN solutions as a pre-cursor gained by the FD algorithm are important, but can *not* represent the whole data set.

Voting and Moral Reasoner problem domains: The FD algorithm does *not* predict any functional dependencies among attributes in the Voting and Moral Reasoner data sets. The pruning algorithm applied after the ANNs' training, however, eliminates significant numbers of input nodes. The cascade network solution of the Voting data set after pruning contains only 11 attributes out of the 16 attributes, keeping all the important attributes such as *Water-project-cost-sharing, Budget resolution, Physician fee, Missile, Immigration, Crime, etc.* The BpTower ANN solution keeps all these 11 attributes and adds *Education spending, and Infant-handicapped* as well (Table 6.8). Similarly the BpTower ANN solution for the Moral Reasoner data set elicits the attributes that are recognized by the cascade network solution, with an additional couple of attributes (Table 6.9).

Cleveland heart disease problem domain: The FD algorithm predicts *3 subsets of attributes* each containing 11 attributes out of 13 attributes in the discretised Cleveland heart disease data set. All 3 combinations of attributes exclude the *Fasting blood sugar* attribute from the solutions. The pruning algorithm for both the cascade and BpTower network solutions also found the *Fasting blood sugar* attribute inefficient for further processing. Both the solutions have also considered the *Sex* attribute irrelevant for rule generalization (Table 6.10), agreeing with one of the subsets recognized by the FD algorithm.

Breast Cancer problem domain: The FD algorithm predicts *19 subsets* of functionally dependent attributes in the Breast Cancer data set, each containing 4 or 5 out of 9 attributes. The pruning algorithm does not eliminate a whole attribute (sparse-coded) but rather recognizes the effective values of each attribute. The input space for both the cascade and BpTower solutions has gone down considerably from *90 to 42*. Both the solutions agree for most of the input space except for minor variations in the values. Table 6.11 shows the performance of ANNs after pruning.

Queensland Rail problem domain: The FD algorithm does *not* elicit existing functional dependencies among attributes in the QR problem domain. On the other hand, the pruning algorithm reduces half of the input space (Table 6.12) in the ANNs built by the cascade and BpTower learning algorithms. The important values of the attributes were distinguished. The level crossing problem domain contains discrepancies in data caused by differing coding schemes. GYAN phase2 is able to recognize such attributes and finds them irrelevant to keep for further processing. For example, the *Sign* attribute was found redundant, as the *sign* value of the *Protection* attribute was considered more effective. The *Light* attribute was not included in the final set of attributes because of the presence of the *Traffic light* attribute. Furthermore the information about *Sleepers* and the *Orientation of trains* was found to be unimportant.

6.2.2 Phase2: Summary and conclusion

GYAN phase2 employs the pruning algorithm to arrive at an ANN solution which requires the minimum number of input nodes and links only to represent the given problem after GYAN phase1 (training of cascade and BpTower ANN) is completed. The following conclusions can be drawn based on the above experiments.

- The important outcome of applying the pruning algorithm to the ANN solution is that irrelevant input values in the instance space are signif-

icantly reduced. Only the important parts of the problem domain that are effective for the ANN's predictions remain.

- A salient feature of GYAN phase2 is recognition of the consistent and collective dominance of the same attributes/values in a given problem domain for both types of neural learning (cascade and BpTower).

- Very often, *training with pruning* results in a better (less complex) ANN than *training without pruning*, and only rarely results in worse ANNs. This agrees with the findings of [Prechelt, 1995].

- In general a *small distance* is required to group the weight states into a reasonable number of clusters in the constrained cascade and BpTower network solutions. The reason is that the restrictions imposed upon ANNs during training (GYAN phase1) encourage the building of ANNs with weights clustered in small intervals. When the distance between clusters is set to a large value (more than 3), each cluster normally contains large number of elements which results in no reduction. The cumulative effect of such clusters is higher than that of clusters with a lesser number of elements, towards the ANN's prediction.

- The distance metric is usually *smaller* in the ANNs produced by the *BpTower algorithm than in those produced by the cascade-correlation algorithm.* The reason is that the cascade-correlation training algorithm produces a ANN with more distinct weights than BpTower. The number of weight clusters is greater in BpTower solutions than in the cascade solutions, but each cluster contains a lesser number of elements. As a result, the pruned BpTower ANN usually contains more elements than the pruned cascade ANN.

- The advantage of reducing the input space after the ANN training on the original data set (relative to reducing the input space before neural learning) can be clearly seen with the Mushroom problem domain. In this domain, the functional dependency algorithm restricts the input

space to the specific subset of attributes to be presented to the ANNs. The pruning algorithm is then able to eliminate a few more input nodes from the ANNs, approximating a *subset* of the problem. However, when the original set of attributes is presented, the trained ANNs are able to find relationships among all the attributes. The pruning algorithm now eliminates a significant number of superfluous input nodes corresponding to attributes/values in the problem domain after ANN learning, and determines all the important features for representing the data set as an entire set of rules.

- It is not essential that the FD algorithm find functional dependencies among the attributes of every data set to which it is applied. The pruning algorithm, however, reduces the hypothesis space for every data set (except the forest remote sensing problem).

- Analysis of weight clusters shows that the individual weight elements in a cluster rarely show the required relationship (dependencies) among themselves. For example, none of the clusters in the Monk1 problem domain could show the logical relationship of *Head_shape* = *Body_shape* among its members. The non-dependencies among elements in a cluster show the need of applying further analysis to the remaining weights or reduced size networks in order to represent the embedded knowledge in the form of symbolic rules.

6.3 Gyan applied in phase3

This section evaluates the extracted rule sets from the ANNs trained on different data sets. GYAN phase3 provides a number of choices for generating predicate rules from a trained ANN. The availability of several propositional rule-extractors allows GYAN to make a selection according to size of the databases, neural architectures, and requirements for the extracted rules (such as accuracy, comprehensibility, etc). The decompositional rule extraction technique *LAP* and the pedagogical rule extraction technique *RuleVI* are applied to ex-

tract propositional rules from the cascade and BpTower ANNs. The *RULEX* technique is applied to extract rules from the trained CEBPNs. The successful mapping of propositional rules, derived from the propositional extractors, to predicate rules also assures wide utilization of the developed algorithm in order to transform propositional rules into predicate rules. In this section, the performance of various algorithms used in GYAN phase3 is evaluated against the set of criteria listed in the preceding chapter.

The extracted predicate rules are compared against those produced by the FOIL system [Quinlan, 1990]. FOIL forms a good basis for comparison with GYAN. In both systems (1) ideas of both propositional and relational learners are incorporated, and (2) the classification rules are expressed in a restricted form of first-order logic *i.e.* function-free predicates. The expressiveness of the rules generated by FOIL however is better than that of GYAN as the former system supports iteration and recursion in its representation.

GYAN is fundamentally very similar to the multi-stage LINUS system [Lavrac *et al.*, 1991]. Within the LINUS system, concept descriptions are induced by propositional learners and then if-then rules are transformed into the form of typed non-recursive Horn clauses with negation. GYAN can also be compared with LINUS but the implemented version of this system was not available for comparison.

The C4.5 [Quinlan, 1993a] decision tree learning algorithm is a well established program to learn propositional rules from data. C4.5 is also applied to the data sets, as the rules extracted by this decision tree learner can also be extended to predicate rules by the generalization algorithm (chapter 4, Figure 4.9).

6.3.1 Symbolic representation of ANN solutions

The main focus of GYAN is to transform the knowledge embedded in trained ANNs into predicate rules. However, the component program to generate predicate rules from propositional rules in the methodology can be applied to any propositional learner.

Tables 6.14 to 6.35 report the rule-extraction performance on the data sets discussed in chapter 5. The classification (class'n) mismatch is given as the ratio of instances that are incorrectly classified to the total number of instances (training or testing). Fidelity of a rule extraction method is a measure of the agreement between the ANN and the extracted rule set. Rule comprehensibility is reported as the number of antecedents (antcd's), the number of propositional rules (including the number of positive and negative expressions), the number of predicate (pred') rules (including the number of positive and negative rules) generated from the propositional rules. The number of CNF expressions extracted using the *LAP* algorithm includes the expressions for the output node and hidden nodes. FOIL generates positive rules only. Some of the predicate rules generated using GYAN are listed in appendices.

Monk1 problem domain: The objective of the Monk1 problem domain is to recognize robots which are monks based on the two existing rules: (1) *Head_shape* = *Body_shape*, or (2) *Jacket_color* = *red*. GYAN phase3 using the *RuleVI* propositional technique does not extract 100% accurate rules because of the presence of the attribute *Has_tie* in the extracted rule set for the cascade-correlation network (Table 6.14). GYAN phase2 removes the irrelevant attributes/values, and the extracted rule set now contains 100% accurate rules. Similarly the rule set extracted by the *RuleVI* technique for the Bp-Tower solution misclassifies some of the patterns due to the presence of the irrelevant attributes *Is_smiling, Holding, and Has_tie*. GYAN phase2 removes all the irrelevant attributes, and the resulting rule set is able to classify all the patterns correctly.

Table 6.14: Rule-extraction performance on the Monk1 data set. The original data set contains the 432 (124 true + 308 false) instances. When the input domain is 3 attributes containing only 8 (3+3+2) values, then the total number of instances is 18 (3*3*2) (in the case of the pruned cascade-correlation network). When the input domain is 3 attributes containing 10 (3+3+4) values, then the total number of instances is 36 (3*3*4) (in the case of the Pruned BpTower network).

Gyan using	Class'n Mismatch		Fidelity (%)
	Training	Testing	
CC-Lap	0/124	0/308	100
PCC-Lap	0/18	0/18	100
CC-RuleVI	0/124	12/308	97.22
PCC-RuleVI	0/18	0/18	100
BT-Lap	0/124	0/308	100
PBT-Lap	0/36	0/36	100
BT-RuleVI	0/124	46/308	89.35
PBT-RuleVI	0/36	0/36	100
Rulex	0/124	0/308	100
C4.5	0/124	0/308	-
Foil	0/124	0/308	-

The predicate rule-set representation of the cascade network utilizing the *RuleVI* algorithm contains one negative rule stating that the first two attributes contain two different values (variables), and two positive rules stating that the *Jacket_color* is *red* or that the first two attributes contain the same values (variables). During the mapping from propositional to predicate rules, compatible predicates (predicates with same symbol and sign) are only replaced by the least general values. The conjunctive expressions extracted by *LAP* do not contain many compatible predicates during the mapping procedure. As a result, the number of predicate rules generalized from the propositional rules is less for the *RuleVI* algorithm than for the *LAP* algorithm. Also, the rule-set for GYAN phase3 utilizing the *LAP* algorithm contains rules derived from the single hidden node, in addition to rules derived from the output node. The rules for the cascade solution clearly state that the hidden node learns *Jacket_color* = *red* or *Head_shape* = *round* $\wedge$ *Body_shape* = *round*, indicating low output but producing high output for the output node, which

Table 6.15: Comprehensibility of rules for the Monk1 data set. Rule comprehensibility is reported as the number of antecedents, the number of propositional rules (including the number of positive and negative expressions), the number of total entities appearing in the final rule set and the number of predicate rules generated from the propositional rules.

Gyan using	No of Antcd's	No of Conj've Expressions	No of Entities	No of Pred' Rules
CC-Lap	63	18 (8,10)	10	16 (6,10)
PCC-Lap	45	18 (8,10)	8	16 (6,10)
CC-RuleVI	61	22 (5,17)	12	3 (2,1)
PCC-RuleVI	19	10 (4,6)	7	3 (2,1)
BT-Lap	42	18 (9,9)	10	16 (7,9)
PBT-Lap	42	18 (9,9)	10	16 (7,9)
BT-RuleVI	164	45 (27,18)	16	8 (6,2)
PBT-RuleVI	72	25 (8,17)	10	4 (3,1)
Rulex	7	4 (4,0)	7	2 (2,0)
C4.5	7	4 (4,0)	7	2 (2,0)
Foil	-	-	7	2 (2,0)

also learns the rest of the target concept. On the other hand, the rules for the BpTower network solution clearly state that: when *Head_shape* = *square* $\wedge$ *Body_shape* = *square* or *Head_shape* = *octagon* $\wedge$ *Body_shape* = *octagon* then the hidden node produces high output and generating high activation for the output node that learns the rest of the target concept. The hidden node in the BpTower solution also learns the inverted values of the attribute *Jacket_color* *i.e.* the hidden node produces low output when the *Jacket_color* is *yellow or green or blue.* Table 6.15 shows that the number of antecedents in conjunctive expressions is significantly reduced with GYAN phase2, that successfully reduces the input space of ANNs trained in GYAN phase1. The predicate rules extracted by FOIL and C4.5 (mapped by the generalization algorithm) are the same as the positive predicate rules extracted by GYAN phase3 utilizing *RuleVI* for the cascade network solution.

Monk2 problem domain: The objective of the Monk2 problem domain is to recognize the robots as monk if only two of the six attributes have their first value. If more than two attributes or less than two attributes contain

Table 6.16: **Rule-extraction performance on the Monk2 data set. The training and test instances are processed separately to reduce the input dimensionality in Gyan phase2. The compressed input space includes the first value of all the attributes, with the rest of the values for each attribute being identified by a not-first value.**

Gyan using	Class'n Mismatch		Fidelity (%)
	Training	Testing	
CC-Lap	0/169	1/263	100
PCC-Lap	0/52	1/60	100
CC-RuleVI	0/169	74/263	83.1
PCC-RuleVI	0/52	6/60	95.54
BT-Lap	0/169	1/263	100
PBT-Lap	0/52	1/60	100
BT-RuleVI	0/169	100/263	77.08
PBT-RuleVI	0/52	6/60	95.54
Rulex	0/169	58/263	98.14
C4.5	29/169	110/263	-
Foil	29/169	111/263	-

Table 6.17: **Comprehensibility of rules for the Monk2 data set. The predicate rules utilizing *LAP* contain rules from the single hidden node as well as from the output node.**

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-Lap	284	63 (36,27)	17	58 (31,27)
PCC-Lap	203	63 (36,27)	12	58 (31,27)
CC-RuleVI	750	130 (60,70)	17	29 (15,14)
PCC-RuleVI	188	38 (15,23)	12	28 (15,13)
BT-Lap	300	63 (31,32)	17	58 (31,27)
PBT-Lap	190	63 (36,27)	12	58 (31,27)
BT-RuleVI	765	138 (64,74)	17	30 (15,15)
PBT-RuleVI	185	38 (15,18)	12	28 (15,13)
Rulex	63	15 (15,0)	17	15 (15,0)
C4.5	35	11 (3,8)	17	-
Foil	-	-	17	18 (18,0)

their first value then an instance must not be categorized as a monk. The ANNs trained for the Monk2 problem domain have learned the above relationship except when all the six attributes have their first values. Consequently the rule sets extracted using the decompositional technique *LAP* for both the ANNs fail to classify that particular instance. With the removal of a few redundant nodes from the input space of each of the ANN, the accuracy of the extracted rule sets using the *RuleVI* pedagogical rule-extraction technique has considerably increased (Table 6.16).

The minimum number of rules to represent the Monk2 problem domain is 15, where each rule contains the first value for exactly two attributes and the negated first value for the rest of the attributes. If negation of the attributes is not permitted in the rule expressions then the total number of rules obtained by enumerating all of the possible permutations over the counterpart of negated antecedents is 142. The conjunctive expressions extracted by the *RuleVI* algorithm are reduced from 60 to 15 predicate rules representing the positive target concept. With the removal of redundant nodes (corresponding to values other than the first value for each attribute) there is significant reduction in the number of antecedents (Table 6.17).

The rules utilizing the *LAP* algorithm for the cascade and BpTower solutions clearly state that the output node learns part of the positive concept *i.e.* any two attributes have their first values and the hidden node learns the remaining part of the concept *i.e.* the rest of the four attributes do not have their first values. The hidden node in the BpTower solution learns the remaining part of the concept by producing high activation to the output node, while the hidden node in the cascade ANN solution learns the remaining part of the concept by producing low activation to the output node.

The accuracy of the predicate rules obtained by the FOIL symbolic learner is quite low (67%) compared to GYAN. Also the number of rules generated

by FOIL is greater than the number of (positive) predicate rules extracted by GYAN with *RuleVI* and *RULEX*.

Monk3 problem domain: The objective of the Monk3 problem domain is to recognize which robots are monks based on the two existing rules: (1) $Body\_shape = \neg octagon \wedge Jacket\_color = \neg blue$ and (2) $Holding = sword \wedge Jacket\_color = green$. Due to the (class/target) noise present in the training data, the trained ANNs do not classify patterns 100% accurately. Consequently, the predicate rules utilizing the decompositional algorithm *LAP*, which has the ability to extract rules with high fidelity, also contain some classification error (Table 6.18). Even after the removal of superfluous input nodes from the cascade network, the inconsistency in one of the training patterns remains present. The inconsistent pattern consists of $Body\_shape = octagon \wedge Holding = sword \wedge Jacket\_color = blue$ that contradicts the target concept. The BpTower networks are able to distinguish the noise present in the training patterns better than cascade networks. BpTower misclassifies the above mentioned pattern, in addition it incorrectly classifies two more patterns as monks consisting of $Body\_shape = round \wedge Holding = sword \wedge Jacket\_color = green$, and $Body\_shape = square \wedge Holding = sword \wedge Jacket\_color = green$, patterns consisting of attributes mapping to the incorrect target concept. In fact, BpTower is able to predict the correct target concept and the noise present in the patterns by classifying them incorrectly.

GYAN phase3 utilizing *RuleVI* for the pruned cascade ANN solution extracts two positive rules (the same as the target rules) and two negative rules exclusively mentioning that the $Body\_shape = octagon \wedge Jacket\_color = blue$ or the $Body\_shape = octagon \wedge Holding = \neg sword$. The cascade network after pruning does not need a hidden node to learn the Monk3 concept. As a result, similar predicate rules are generated using *LAP* and *RuleVI*. The BpTower solution has three important attributes and an additional attribute *Has_tie* after applying GYAN phase2. The *Has_tie* attribute does not appear in the predicate

Table 6.18: Rule-extraction performance on the Monk3 data set

Gyan using	Class'n Mismatch		Fidelity (%)
	Training	Testing	
CC-Lap	1/122	20/310	100
PCC-Lap	1/24	0/25	100
CC-RuleVI	1/122	60/310	90.7
PCC-RuleVI	1/24	0/25	100
BT-Lap	3/122	6/310	100
PBT-Lap	3/62	0/74	100
BT-RuleVI	3/122	99/310	78.24
PBT-RuleVI	3/62	1/74	98.64
Rulex	7/122	9/310	100
C4.5	6/122	10/310	-
Foil	6/122	65/310	-

Table 6.19: Comprehensibility of rules for the Monk3 data set

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-Lap	231	48 (23,25)	12	43 (22,21)
PCC-Lap	15	5 (2,3)	9	4 (2,2)
CC-RuleVI	141	39 (15,24)	12	18 (8,10)
PCC-RuleVI	25	13 (7,6)	9	4 (2,2)
BT-Lap	470	92 (46,46)	17	67 (35,32)
PBT-Lap	15	5 (2,3)	9	4 (2,2)
BT-RuleVI	254	59 (34,25)	17	9 (4,5)
PBT-RuleVI	59	21 (13,8)	9	4 (2,2)
Rulex	6	2 (2,0)	6	2 (2,0)
C4.5	25	12 (7,5)	11	-
Foil	-	-	11	13 (13,0)

Table 6.20: Rule-extraction performance recognizing a water area in the Remote Sensing problem domain

Gyan using	Class'n Mismatch		Fidelity
	Training	Testing	(%)
CC-Lap	0/85	0/76	100
PCC-Lap	0/75	0/68	100
CC-RuleVI	0/85	0/76	100
PCC-RuleVI	0/75	0/68	100
BT-Lap	0/85	0/76	100
PBT-Lap	0/75	0/68	100
BT-RuleVI	0/85	0/76	100
PBT-RuleVI	0/75	0/68	100
Rulex	0/85	0/76	100
C4.5	3/85	3/76	-
Foil	0/85	0/76	-

rule set utilizing the *LAP* algorithm, and rule-comprehensibility is equivalent to that of the cascade solution. Due to presence of this attribute, GYAN phase3 utilizing *RuleVI* extracts a larger number of conjunctive expressions than its counterparts (Table 6.19). During the mapping from propositional to predicate rules, this redundant attribute is removed and a competitive rule set is generated.

The accuracy of the predicate rules obtained by the FOIL symbolic learner is significantly lower than that of the rule sets generated using *LAP* and *RuleVI* for the ANN solutions after pruning. The rule-set generated by FOIL fails to reflect the required relationship among the attributes and contains many dispensable rules.

Remote Sensing problem domain: GYAN phase3 generates 100% accurate sets of predicate rules from all the combinations of algorithms for the Remote Sensing problem domain (water and forest - Tables 6.20, 6.21), although the issue of generalization needs to be addressed here. The Remote Sensing problem domain, consisting of 3 attributes each having 5 values, yields an input space with 125 distinct values. The training set consists of only 85 unique instances, and does not cover the entire input space. The problem of

Table 6.21: Rule-extraction performance recognizing a forest area in the Remote Sensing data set

Gyan using	Class'n Mismatch		Fidelity (%)
	Training	Testing	
CC-Lap	0/85	0/76	100
CC-RuleVI	0/85	0/76	100
BT-Lap	0/85	0/76	100
BT-RuleVI	0/85	0/76	100
Rulex	0/85	0/76	100
C4.5	5/85	4/76	-
Foil	0/85	0/76	-

Table 6.22: Comprehensibility of rules extracted for the Remote Sensing data set recognizing water areas

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-Lap	43	9 (3,6)	15	7 (1,6)
PCC-Lap	37	9 (3,6)	13	7 (1,6)
CC-RuleVI	31	16 (4,12)	14	7 (1,6)
PCC-RuleVI	22	12 (4,8)	13	5 (1,4)
BT-Lap	45	9 (3,6)	15	7 (1,6)
PBT-Lap	37	9 (3,6)	13	7 (1,6)
BT-RuleVI	26	14 (4,10)	14	7 (1,6)
PBT-RuleVI	24	12 (4,8)	13	7 (1,6)
Rulex	9	2 (2,0)	7	1 (1,0)
C4.5	2	1 (1,0)	2	-
Foil	-	-	6	4 (4,0)

generalization becomes more apparent for the problem of recognizing water areas, for which the supporting instances are very few.

The *RULEX* algorithm extracts rules from trained CEBP networks, which use local response units to learn the target concepts. The *RULEX* algorithm extracts rules with high fidelity to the CEBPN. But the CEBPN is constructed by responding to regions that fit the training data equally. A CEBPN solution is not acceptable which makes generalizations about regions of the input space for which there is no supporting evidence in the training set. Consequently, the generated rule-set utilizing *RULEX* only contains antecedents that are positive training instances in the problem space for water recognition *i.e.* *Red* =

Table 6.23: Comprehensibility of rules extracted for the Remote Sensing data set recognizing forest areas

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-Lap	53	10 (5,5)	15	5 (3,2)
CC-RuleVI	149	55 (15,40)	15	4 (3,1)
BT-Lap	55	10 (5,5)	15	5 (3,2)
BT-RuleVI	163	67 (20,47)	15	3 (2,1)
Rulex	19	3 (2,1)	12	3 (2,1)
C4.5	16	10 (5,5)	12	-
Foil	-	-	12	13 (13,0)

very_low or low $\wedge$ *Green = medium or high* $\wedge$ *Blue = high or very_high.*

The *RuleVI* algorithm utilizes training patterns to extract rules from ANNs. The extracted rule set very much depends on the contents and the quality of the training set. As a result, the generated positive rules utilizing *RuleVI* are specific to the training set. The extracted rule set contains the same antecedents as *RULEX* for water recognition problem.

In contrast, the extracted rule sets utilizing the *LAP* algorithm are derived from analysis of weight parameters in the underlying trained ANNs, and are independent of any direct effect from the training patterns. The *LAP* algorithm is guaranteed to extract rules with 100% fidelity from a perceptron (or ANNs employing a hard-limit threshold function in non-input units) [Hayward *et al.*, 1996]. Consequently, the generated positive rules utilizing *LAP* contain generalized Antecedents that are not specific to the training instances in the problem space for water recognition. The resulting positive rules contain the antecedents *Red = very_low or low or medium or high* $\wedge$ *Green = very_low or low or medium or high* $\wedge$ *Blue = high or very_high.* In other words, the predicate rule-set utilizing the *LAP* algorithm contains more generalized rules than others (Table 6.22).

35% of the training instances in the Remote Sensing problem domain support the target class of forest. Consequently, the predicate rule-sets gener-

Table 6.24: Rule-extraction performance on the Mushroom data set

Gyan using	Classification Mismatch		Fidelity (%)
	Training	Testing	
PCC-Lap	0/42	0/42	100
CC-RuleVI	0/2708	128/5416	98.42
PCC-RuleVI	0/42	0/42	100
CC-Lap (FD)	0/163	0/103	100
PCC-Lap (FD)	0/81	0/28	100
CC-RuleVI (FD)	0/163	0/103	100
PCC-RuleVI(FD)	0/81	0/28	100
PBT-Lap	0/54	0/54	100
BT-RuleVI	0/2708	256/5416	96.84
PBT-RuleVI	0/54	0/54	100
BT-Lap (FD)	0/163	0/103	100
PBT-Lap (FD)	0/48	0/48	100
BT-RuleVI(FD)	0/163	0/103	100
PBT-RuleVI(FD)	0/48	0/48	100
Rulex	14/2708	32/5416	99.95
C4.5	0/2708	8/5416	-
Foil	0/2708	10/5416	-

ated from all possible combinations of algorithms elicits the same set of values/attributes. The positive instances are basically grouped into two regions (*Red = low or medium or high or very_high* $\wedge$ *Green = low or medium* $\wedge$ *Blue = high or very_high*) and (*Red = low or medium or high* $\wedge$ *Green = very_low or low* $\wedge$ *Blue = very_low or low or medium*). All of the predicate rule-sets show this behavior (Table 6.23).

Mushroom problem domain: The mushroom problem domain contains 22 attributes and the problem space can have 10^{14} possible combinations of the attribute values. It is not feasible for the *LAP* algorithm to iterate exhaustively through such a large potential weight space to represent the ANN as symbolic rules with fidelity of 100%. GYAN phase2 has shown the presence of many superfluous attributes/values. The inaccuracy present in the rule-set generated for the entire problem domain (*RuleVI* and *RULEX*) (table 6.24) also indicates the existence of irrelevant attributes/values and a high level of redundant data. With the application of GYAN phase2, the algorithms are

Table 6.25: Comprehensibility of rules for the Mushroom data set

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
PCC-RuleVI	73	24	26	11 (5,6)
PCC-Lap (FD)	373	44	21	18 (8,10)
CC-RuleVI (FD)	400	92	39	8 (2,6)
PCC-RuleVI(FD)	73	24	19	7 (2,5)
PBT-RuleVI	97	26	23	17 (7,10)
PBT-Lap (FD)	413	36	19	28 (13,15)
BT-RuleVI (FD)	669	125	40	10 (3,7)
PBT-RuleVI (FD)	29	11	16	6 (3,3)
Rulex	90	3	61	3
C4.5	23	15 (9,6)	19	-
Foil	-	-	14	6 (6,0)

able to extract rules with 100% accuracy and fidelity.

The *RULEX* algorithm extracts the single most important rule for edible mushroom, *Odour = almond or anise or none* covering 98.5% of the instances in the problem domain, if the training of the CEBPN is stopped at an early stage. Extended training to know more about the data set results in increased accuracy, poor comprehensibility and two more rules for the poisonous mushrooms including a total of 20 attributes (Table 6.25).

This raises the issue of totality of knowledge versus the comprehensibility of the generated rule-set. The accomplishment of the totality of the knowledge embedded within a trained ANN adversely affects the comprehensibility of rules. Tickle [1998] has also observed that only a subset of the total set of extracted rules (required to classify the set of data used to train the ANN) may actually express the knowledge within the ANN initially. For example, there is a trade-off between accuracy and comprehensibility of the rule-sets extracted by the *RULEX* algorithm. When the local hidden nodes do not cover the whole instance space a more comprehensible rule-set is generated but with less accuracy (covering less instances). This is one reason why *LAP* and *RuleVI* algorithms extract more rules than other competitive algorithms

Table 6.26: Rule-extraction performance on the Voting data set

Gyan using	Classification Mismatch		Fidelity (%)
	Training	Testing	
PCC-Lap	1/158	9/79	100
CC-RuleVI	1/261	27/174	94.91
PCC-RuleVI	1/158	15/79	97.46
PBT-Lap	0/169	10/84	100
BT-RuleVI	0/261	67/174	85.51
PBT-RuleVI	0/169	20/84	96.04
Rulex	0/261	7/174	100
C4.5	4/261	11/174	-
Foil	5/261	16/174	-

as they consider the whole space approximated by ANNs.

The predicate rule-set utilizing the *RuleVI* algorithm for the cascade ANN solution elicits 200 rules for the entire problem domain, whereas the number of predicate rules for the pruned cascade network utilizing *RuleVI* is only 11 and for the ANN trained on the subset of attributes obtained by the function dependency algorithm it is only 8. Interfacing with the knowledge base reasoner helps to solve this issue by providing a user interface whenever an understanding is required. A better option is to reduce the hypothesis space before or after, a problem is presented to the ANN. There is no significant difference in the accuracy and comprehensibility of the rule-sets generated by different component algorithms.

Voting problem domain: The Voting problem domain comprises 16 attributes, each of them having three possible values and resulting in 10^7 possible combinations of the attribute values. DNF expressions were not successfully extracted for the component *LAP* rule-extraction algorithm because of its exponential algorithmic complexity. For the reduced instance space, the predicate rule-sets employing the *LAP* algorithm generate rules with 100% fidelity (Table 6.26). The number of misclassified patterns is reduced after applying GYAN phase2.

Table 6.27: Comprehensibility of rules on the Voting data set. Details of some of the rule-sets are not reported because of the poor comprehensibility.

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
PCC-Lap	839	336 (168,168)	24	119 (65,54)
CC-RuleVI	577	80 (40,40)	39	68 (34,34)
PCC-RuleVI	120	30 (15,15)	21	10 (6,4)
PBT-Lap	913	363 (170,193)	28	121 (66,55)
BP-RuleVI	609	86 (42,44)	41	70 (36,34)
PBT-RuleVI	300	65 (33,32)	28	42 (28,14)
Rulex	17	4 (4,0)	15	4 (4,0)
C4.5	7	4 (1,3)	6	-
Foil	-	-	14	6 (6,0)

All the predicate rule-sets contain the most important rule consisting of the attributes *Physician fee* = *no* and *Adapt_budget_resolution* = *no* to support the democrat vote. They also contain a rule such as *Mx_missile* = $\neg$*no* and *Immigration* = *no*. For this domain the *RULEX* algorithm and the symbolic rule-induction algorithms extract fewer rules with comparable accuracy as compared to the rule-extraction techniques *LAP* and *RuleVI* (Table 6.27). The extracted conjunctive expressions are further generalized into predicate rules resulting in an even smaller number of rules.

The *RULEX*, C4.5 and FOIL techniques generate rules by partitioning data (*RULEX* using local response units, C4.5 and FOIL according to the gain ratio) and then search locally for important rules. The *LAP* and *RuleVI* techniques generate rules by adopting a distributive approach *i.e.* consider and test each attribute according to the ANN's response. The *LAP* algorithm accomplishes this by analyzing weights for each node in the ANN and *RuleVI* does so by analyzing the effect of each input node on the ANN's response by changing the input value of a sample pattern. In other words, (1) all the attributes are exhaustively tested to determine their effect on the ANN's response and (2) if an attribute is found to contribute, it is included in rules irrespective of the ranking or number of patterns correctly classified by the extracted rule.

Table 6.28: Rule-extraction performance on the Moral Reasoner data set

Gyan using	Classification Mismatch		Fidelity (%)
	Training	Testing	
CC-Lap	0/162	0/40	100
PCC-Lap	0/151	0/40	100
CC-RuleVI	0/162	3/40	98.51
PCC-RuleVI	0/151	0/40	100
BT-Lap	0/162	0/40	100
PBT-Lap	0/160	0/40	100
BT-RuleVI	0/162	15/40	92.57
PBT-RuleVI	0/160	10/40	95
Rulex	82/162	20/40	50.5
C4.5	0/162	0/40	-
Foil	0/162	4/40	-

Table 6.29: Comprehensibility of rules for the Moral Reasoner data set

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-RuleVI	233	43 (21,22)	28	35 (18,17)
PCC-RuleVI	253	49 (24,25)	25	38 (18,20)
BT-RuleVI	1103	116 (58,58)	44	79 (30,49)
PBT-RuleVI	367	62 (31,31)	30	35 (12,13)
Rulex	11	4 (2,2)	10	4 (2,2)
C4.5	13	6 (9,6)	10	-
Foil	-	-	10	2 (2,0)

Thus the information gathered by *LAP* and *RuleVI* algorithms is sometimes excessive but is able to express the essence of learning in ANNs.

Moral Reasoner problem domain: The accuracy and fidelity of the predicate rule-set for the Moral Reasoner problem domain employing *RULEX* is very low compared to other algorithms (Table 6.28). This occurs because of the *'corner effect'* in *RULEX* [Andrews and Geva, 1994]. Essentially *RULEX* approximates the hyper-ellipse obtained by the summation of all the ridges, according to input dimensions for a local hidden node, by a hyper-rectangle during the formation of the rule-set. This leaves a space for the point within the hyper-rectangle that will be classified as belonging to the rule, but is not

Table 6.30: Rule-extraction performance on the Cleveland heart disease data set

Gyan using	Classification Mismatch		Fidelity (%)
	Training	Testing	
CC-RuleVI	1/216	39/87	92.1
PCC-RuleVI	1/216	28/77	93.63
BT-RuleVI	1/216	50/87	87.78
PBT-RuleVI	4/206	39/77	88.69
Rulex	42/216	29/87	78.81
C4.5	30/216	22/87	-
Foil	0/216	15/87	-

actually classified by the ANN (as it is outside the area of the hyper-ellipse). Clearly there is scope for *false positive* errors *i.e.* points (instances) that do not belong to the target class but which the extracted rule-set classifies as members of the target class, and *false negative* errors *i.e.* points that belong to the target class but which are classified as negative instances by the rule. Analysis of the *RULEX* results show that all the inaccuracy errors are *false negatives i.e.* the points are correctly classified as positive instances by the ANN but the extracted rule-set classifies them as negative instances (recalling that the RMSE of test set is quite high, Figure 6.9).

All the predicate rule-sets contain the important attributes such as *Benefit-victim = no*, *Severity-harm = yes*, *Intervening-contribution = no* and *External-force = no* to show that the person is found guilty of harm-doing. The predicate rule-sets utilizing the *LAP* and *RuleVI* algorithms contain some additional attributes such as *Mental-state = reckless, negligent or intended*, *Sufficient-for-harm = yes*, *Necessary-to-harm = no, etc.* (Table 6.29). The predicate rule-set generated by GYAN agrees with the given 'Horn-clause theory'.

Cleveland heart disease problem domain: The Cleveland data set consists of mixed (continuous + discrete) attributes. The *LAP* and *RuleVI* algorithms were applied to the ANNs trained for the modified data set in which the continuous values were discretised. The *RULEX* algorithm was applied to

Table 6.31: Comprehensibility of rules for the Cleveland heart disease data set

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-RuleVI	1221	161	36	121 (50,71)
PCC-RuleVI	1020	136	32	98 (35,63)
BT-RuleVI	2079	185	37	133 (64,69)
PBT-RuleVI	1324	159	33	110 (34,76)
Rulex	47	4	30	4 (2,2)
C4.5	23	8 (2,6)	18	-
Foil	-	-	19	13 (13,0)

Table 6.32: Rule-extraction performance for the Breast Cancer data set

Gyan using	Classification Mismatch		Fidelity (%)
	Training	Testing	
CC-RuleVI	0/411	75/272	90.04
PCC-RuleVI	0/205	22/134	94.39
BT-RuleVI	0/411	156/272	78.47
PBT-RuleVI	0/223	37/150	91.15
Rulex	1/411	13/272	99.4
C4.5	9/411	18/272	-
Foil	0/411	22/272	-

the CEBPN trained on the original data set. The accuracy of the resulting predicate rule-sets is better when the modified data set is used to generate them than when the original data set is used (Table 6.30, 6.31). The intervals obtained by the propositional rules using *RULEX* and C4.5 agree with the categories used in sparse-coding for the cascade and BpTower networks. There is no support for numerical operations performed in SHRUTI. The queries are answered by matching the facts rather than by using any numerical calculation. The intervals revealed by the conjunctive expressions obtained using the *RULEX* algorithm are considered as categorized attributes in the generation of the predicate rule-set and as further sub-concepts in the automated knowledge base.

Breast Cancer problem domain: There is significant improvement in accuracy and comprehensibility of the predicate rule-set generated for the ANN

Table 6.33: Comprehensibility of rules for the Breast Cancer data set

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
CC-RuleVI	2513	340	53	287 (98,189)
PCC-RuleVI	121	46	37	36 (21,15)
BT-RuleVI	2197	274	61	191 (43,148)
PBT-RuleVI	190	60	40	44 (23,21)
Rulex	72	9	36	9 (7,2)
C4.5	22	16 (2,14)	21	-
Foil	-	-	14	11 (11,0)

Table 6.34: Rule-extraction performance on the Queensland Rail data set

Gyan using	Classification Mismatch		Fidelity (%)
	Training	Testing	
PCC-Lap	40/490	4/95	99.5
CC-RuleVI	109/1561	14/173	99.65
PCC-RuleVI	43/490	4/95	100
PBT-Lap	39/500	3/100	99.16
BT-RuleVI	112/1561	16/173	99.48
PBT-RuleVI	44/500	3/100	100
Rulex	109/1561	12/173	99.88
C4.5	92/1561	13/173	-
Foil	120/1561	41/173	-

solutions to the Breast cancer problem domain after applying GYAN phase2 (Table 6.32, 6.33). The *LAP* algorithm is not applied because of the higher combinatorics (10^9) of attribute-values in the problem domain. The analysis of rules obtained by FOIL and C4.5 reveals the presence of two entirely different sets of attributes/values in the generated rule-sets. The 19 subsets of functional dependent attributes determined by the FD algorithm confirms the existence of many associative rules rather than a few potentially dominating attributes in the rules. On the other hand, the predicate rule-sets employing the *LAP* and *RuleVI* algorithms contain most of the possible rules.

Queensland Rail problem domain: The Queensland Rail data set is a real-life problem domain collected from operational data. Because of noise existing in the data, neither of the component programs is able to generate

Table 6.35: Comprehensibility of rules for the Queensland Rail data set

Gyan using	No of Antcd's	No of Conj've expressions	No of Entities	No of Pred' Rules
PCC-Lap	50	27	24	11 (8,3)
CC-RuleVI	202	97	48	18 (10,8)
PCC-RuleVI	50	12	18	6 (4,2)
PBT-Lap	53	28	26	11 (8,3)
BT-RuleVI	237	102	51	20 (13,7)
PBT-RuleVI	56	12	18	6 (4,2)
Rulex	26	1	26	1 (1,0)
C4.5	37	15 (7,8)	26	-
Foil	-	-	15	9 (9,0)

100% accurate predicate rule-sets (Table 6.34). The data set contains very few instances supporting the accident cases. Consequently, there are many *false negative* errors, *i.e.* instances with a target value of 1 (belonging to the target class of *Risky*) but with a rule output of 0 (*i.e.* a *Safe* case). The attributes that appear in rule-sets to state an accident-prone level crossing are: *Protection = nil*, *Road-visibility = poor*, *Train-speed = fast or very-fast (50-160 km/h)*, *Pedestrian = exist*, *Approach-sign = yes*, *etc.* (Table 6.35).

An improvement in the generated rule-sets can be achieved by implementing the concept of voting methods such as 'boosting' [Freund and Shapire, 1997] or 'bagging' [Breiman, 1996] during the learning phase of inductive algorithms. Both bagging and boosting lead to more accurate classifiers at the cost of additional computation by manipulating the training data in order to generate and aggregate different (multiple) classifiers [Quinlan, 1996]. In 'bagging', several classifiers are generated from replicate training sets. Each training set is sampled (with replacements) from the original instances, and is the same size as the original data, with some (random) instances not appearing at all while others appearing more than once. In the Queensland Rail data set, certainly the patterns supporting the *Risky* class will be duplicated in the resampled training sets than the *Safe* ones. In 'boosting', several classifiers are generated from the same (and entire) data. A cost is associated with the patterns that

Table 6.36: The relative overall predictive accuracy of predicate rules

Gyan using		Accuracy (%) Training	Accuracy (%) Testing
LAP	CC	98.11	94.23
	PCC	98.28	95.05
	BT	98.07	94.69
	PBT	98.21	95.15
RuleVI	CC	97.54	83.69
	PCC	97.65	89.57
	BT	97.44	71.65
	PBT	97.59	84.71
Rulex	CEBPN	96.41	89.51
C4.5		96.99	94.05
Foil		97.1	83.98

were misclassified by the previous 'boosting' classifiers, and more attention is paid to these patterns. In the Queensland Rail data set, certainly the patterns supporting the *Risky* class will get higher cost than the *Safe* ones, and the ANNs will learn the features hidden behind these patterns more dominantly than before. In either case (bagging or boosting), the multiple classifiers are combined by voting to form a composite classifier that usually has a higher predictive accuracy than any of its components [Quinlan, 1996].

6.3.2 Phase3: Summary and conclusion

Table 6.36 reports the relative overall performance of predicate rule-sets utilizing different algorithms. The average predictive accuracy is determined by separately measuring the classification accuracy on an unseen pattern set (or training) achieved for each combination of data set and rule-set, and then calculating the average predictive accuracy across all data sets, for each rule set.

The following conclusions can be drawn based on these experiments:

- The accuracy of the generated predicate rules very much depends on the rule-extraction algorithm that has been employed to extract the propositional expressions from the trained ANN.

- The expressiveness of the extracted propositional expressions is enhanced by introducing variables and predicates in rules without loss of accuracy or of fidelity to the ANN solution.

- If a relationship exists among attributes in the data set (in other words the relevance of a particular input attribute depends on the values of other input attributes as in Monk1), then the generalization algorithm is capable of showing that relationship in terms of variables. Otherwise the generalization algorithm first reduces the number of propositional expressions by applying the post-pruning rules and then translates them into predicate form.

- Bergadano and Gunetti [1995] and many others have observed that first-order learning systems (such as FOIL) show a serious degradation of performance when moving from training examples to test data. This agrees with the comparatively lower generalization accuracy obtained by FOIL (Table 6.36). The GYAN methodology avoids this behavior by extracting propositional symbolic information from ANNs first, and then extending the expressiveness of rules into the restricted first-order logic representation by the generalization procedure.

- A transformation of the trained ANN into a meaningful, logical form of rules is required to understand the decision making process of ANNs. The *LAP* and *RuleVI* algorithms convert all the knowledge embedded in ANNs into rules. Sometimes, the comprehensibility of the extracted rule-set is poor and does not serve the intended purpose. In these cases, it becomes necessary to apply a refinement process to remove redundant literals and to generate a more compact form of rules. The mapping of propositional expressions into predicate rules mostly results in a smaller, equivalent sets of rules. This process is especially helpful when a data set contains relational attributes (functional dependent) as in the Monk1 problem domain.

- The comprehensibility of the generated predicate rule-sets using GYAN

(*LAP* and *RuleVI*) is sometimes inferior to those generated by the FOIL system. The fundamental problem is the requirement for explicit negative rules or negated facts in the SHRUTI connectionist rule-based reasoning system for efficient reasoning. GYAN phase4 (SHRUTI) does not consider CWA (closed world assumption) or 'negation by failure'. Essentially, GYAN (using *LAP*) considers the internal architecture of ANNs and explains the ANN in terms of predicates.

- The number of predicate rules generalized from the DNF expressions extracted using the *RuleVI* algorithm is comparatively smaller than those produced by the *LAP* algorithm. During the mapping from propositional to predicate rules, only compatible predicates (predicates with the same symbol and sign) are replaced by least general values. During the rule-extraction phase in the *LAP* algorithm, all possible combinations of attributes are tested against the condition $((|\sum weights| < Bias))$. As a result, fewer compatible predicates are generated after using the *LAP* algorithm. The other reason is that the rule-sets for GYAN phase3 utilizing the *LAP* decompositional algorithm contain rules derived from hidden nodes in the ANNs, in addition to rules derived from the output nodes.

- If the number of attributes is large then a pruning algorithm should be applied to reduce the hypothesis space. Pruning allows the rule extraction methods to consider only the parts of the search space whose elements are those input attributes which are involved in the conjunction of target concepts. A shortcoming of LAP and other decompositional techniques is that these methods become cumbersome when the search space is too large (number of attributes). As the number of dimensions increases, the number of possible combinations of attributes grows exponentially. If all the attributes are not required in the search, the gain is significant. For instance, the search space is reduced by a factor of roughly 2^5 in the simple Monk1 data set. The extracted rule-sets, by RuleVI and other pedagogical methods, only completely 'cover' the set

of instances that are used to obtain them. By removing attributes that do not have any impact on the target concept, the efficiency of such methods is increased. The projection to a space of a lower number of input dimensions by pruning allows translation of the trained ANN into a compact rule set that sometimes seems impossible to obtain from the original high input dimensionality such as the Mushroom problem domain (using *LAP*).

- For all data sets, the number of rules and antecedents is significantly lower in the rule-sets extracted from the pruned ANNs than in the rule-sets extracted from the ANNs before pruning. Based on the results shown in the Tables 6.14 to 6.35, it can be said that the pruning of the ANNs gives an overall better result in rule extraction especially for the data sets involving a large number of attributes and/or values.

- The rules obtained by the different combinations of algorithms are quite different because of the distinct qualities of the associated programs. But they still carry equivalent meaningful information for most of the data sets, as revealed by the classification accuracy and fidelity.

- Implementation of activation functions during the ANN training plays an important role in rule extraction. The cascade network utilizes the sigmoidal activation function when installing hidden nodes. The Bp-Tower network approximates non-input nodes by a Heaviside function after installing. The *RuleVI* implementation employs a sigmoidal activation function when the ANN is used to classify amended instances. The *LAP* implementation approximates each non-input node by a hardlimit threshold function to test the condition $((|\sum max(weights)| < Bias))$. Sometimes the different combination of activation functions creates discrepancies between the ANNs' output and the extracted rules. This problem becomes more apparent when the RMS error of the trained ANN is quite high or the ANN contains many hidden nodes. For example, a hidden node may yield an output of 0.45 for a pattern but is

approximated as 0.0 in BpTower network solutions.

Table 6.36 shows that the combination of the BpTower network and *LAP* yields a better accuracy than the combination of the cascade network and *RuleVI*. Similarly, the *RuleVI* algorithm performs better (extracts more accurate and comprehensible rules) for the cascade network solutions than for the BpTower network solutions.

- The generalization capability of the rule-sets extracted by the *LAP* decompositional technique is better than that of the rule-sets produced by the *RuleVI* pedagogical technique. The *RuleVI* pedagogical technique relies substantially on access to a fixed set of training patterns to generate and test the rules. The *LAP* decompositional technique does not directly utilize training patterns to generate rules, but relies only on the architecture, weights and accuracy of the trained ANNs.

- The *RuleVI* and *RULEX* algorithms are not very noise-tolerant because of the dependence on training data. On the contrary, *LAP* is quite noise-tolerant, as it only relies on the ANN architecture and not on the training patterns.

  The *RuleVI* algorithm generates DNF expressions equivalent to the ANN by sequentially examining responses obtained from the ANN for the query instances (training patterns and modified versions of training patterns). The extracted rule-set very much depends on the contents and the quality of the training set. In general, the rules generated from ANNs utilizing *RuleVI* are *specific to the training set.*

  The *RULEX* algorithm extracts rules from trained CEBPNs which use local response units to learn the target concepts. The *RULEX* algorithm extracts rules with *high fidelity* to the ANN. But the ANN (CEBPN) is constructed by responding to regions that fit the training data equally. A CEBPN solution which makes generalizations about regions of input space for which there is no supporting evidence in the training space, is not acceptable. Consequently, the generated rule-set utilizing *RULEX*

only contains the antecedents that are in the training instances in the problem space.

In contrast, extracted rule-sets utilizing the *LAP* algorithm are (according to the analysis of weight-vectors) representing the underlying trained ANNs and independent of any direct impact of training patterns. The *LAP* algorithm is guaranteed to extract rules with 100% fidelity from a perceptron (or ANNs employing a hard-limit threshold function in non-input units) [Hayward *et al.*, 1996]. Though this quality makes this method very computationally expensive. The classification ability of a rule-set is equivalent to the ANN from which the rule-set is extracted. The extracted rules contain antecedents that are not just specific to the training instances in the problem space but include the generalization capability of the trained ANNs. One of the problems in *LAP* appears to be the approximation of a sigmoidal activation function by a non-input node with a hard-limit threshold function. If the result has many hidden nodes then the previous node may be giving an output of 0.6, but this is considered to be 1 in weight analysis, which may cause a problem.

- If the instances in a data set are inseparable, *RULEX* performs worse than *RuleVI* and *LAP*.

- In general, the comprehensibility of propositional expressions generated by *RULEX* and C4.5 is better than that of *RuleVI* and *LAP*. The generation of rules in *RULEX* (CEBPN) and C4.5 is based on data partitioning. C4.5 uses the split ratio gain to choose the important attributes rather than considering all of them at any instance. The C4.5 algorithm generates a subset of solutions rather than a complete solution for the given problem. This raises the issue of totality of knowledge vs the comprehensibility of generated rule-sets. The extraction of total knowledge embedded within the trained ANNs adversely affects the comprehensibility of the rules. The *LAP* and *RuleVI* algorithms search for rules in the problem space that is approximated by the trained ANN for the

given problem. There is a trade-off between the accuracy and comprehensibility of the rule-set extracted by the *RULEX* algorithm. When the local hidden nodes do not cover the whole instance space, a more comprehensible rule-set is generated but with less accuracy (covering less instances).

- The rule-set extracted from the BpTower network solution, in general, contains a greater number of rules than those obtained from the cascade correlation ANN solution for the same problem. The fundamental reason for this is that the cascade algorithm produces ANNs in which some weights are distinctly larger in magnitude than other weights for the same nodes. This helps a rule-extractor to distinguish important concepts.

- If the underlying trained ANN suffers from overfitting, the rule-extraction performance is degraded. The *RULEX* and *RuleVI* algorithms produce very poorly generalized rules for such ANNs. Basically, the generated rule-sets reflect the quality of the trained ANNs. If an ANN is well-trained and there is not much variation in the test and training cases, then a pedagogical approach is an easy option for generating rule-sets from the trained ANNs.

- Lateral connections among hidden nodes significantly increase the complexity of the process of extracting the knowledge embedded in the underlying ANN using decompositional techniques. Accordingly, the comprehensibility of the extracted rule-set becomes poor and needs to be interfaced with an efficient reasoning system for interactive user explanation.

6.4 Gyan applied in Phase4

Now the remaining task is to integrate the generic and conceptual rules generated from the trained ANNs with an inference engine. Such a system now enables users to pose the queries in order to retrieve the information that they

are interested in. This is also useful in the situations when comprehensibility of the generated rule sets is poor. The rule-sets require some type of post-processing for the user to be able to understand the rules and the decision process of the ANN. The knowledge bases (including quantified rules with predicates, facts and a type-hierarchy) are generated in the syntax recognized by the SHRUTI inference system.

6.4.1 Automated knowledge bases

The GYAN methodology tests the performance (capability of responding queries correctly) of resultant SHRUTI networks generated by initializing automated knowledge bases within the SHRUTI reasoning system. The information about the knowledge bases generated for each problem domain, described in chapter 5, is reported in Tables 6.37 to 6.46.

The SHRUTI reasoning system responds to a query in four possible ways: *'true'; 'false'; 'do not know'; and 'ambiguity in the knowledge base'*. SHRUTI does not assume that failure is negation; that is, it returns *'do not know'* if the query has not been responded to over an extended period of time, asserting that there are no facts to support or deny the query.

Mathematical operators do not have any support in SHRUTI[1], unlike the conventional inference engine Prolog. The representation of common knowledge uses a descriptive fuzzy logic style approach to handle quantities (continuous values) by categorizing them into intervals. The knowledge base generated by utilizing the component *RULEX* algorithm for the Cleveland data base takes the same approach. The decision boundaries of a continuous attribute recognized by the *RULEX* algorithm are treated as sub-concepts in the type-hierarchy.

[1]The current version of SHRUTI supports numerical attribute values and operators such as $=, >, <$

Table 6.37: SHRUTI knowledge base for the Monk1 data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-Lap	3	8	43	16
PCC-Lap	3	8	39	16
CC-RuleVI	4	10	22	4
PCC-RuleVI	3	8	10	4
BT-Lap	3	10	46	18
PBT-Lap	3	10	46	18
BT-RuleVI	6	17	45	9
PBT-RuleVI	3	10	26	5
Rulex	3	8	4	3

Finally, queries are posed to the SHRUTI knowledge base reasoner in order to apply an automated knowledge (facts and rules) to specific circumstances. Some of the queries posed to the SHRUTI networks generated by automated knowledge bases are reported below along with the tables informing about the knowledge bases.

Data base: Monk1 using GYAN (PCC-*LAP*) (Table 6.37)

Query: *?monk1(head_square, body_square, jacket_red)*

Response: *true*

Supporting facts: *output1_pred4(head_square), output1_pred6(body_square), output1_pred0(jacket_red), hidden1_pred1(jacket_red), hidden1_pred1(body_square).*

Data base: Monk1 using GYAN (PCC-*RuleVI*) (Table 6.37)

Query: *?monk1(head_round, X, jacket_red)*

Response: *true*

Supporting facts: *monk1_pred0(head_round, body_round), monk1_pred1(jacket_red).*

Data base: Monk1 using GYAN (PBT-*LAP*) (Table 6.37)

Query: *?monk1(head_round, body_octagon, jacket_yellow)*

Response: *false*

Table 6.38: SHRUTI knowledge base for the Monk2 data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-Lap	6	17	63	65
PCC-Lap	6	12	63	65
CC-RuleVI	6	17	130	30
PCC-RuleVI	6	12	38	29
BT-Lap	6	17	63	65
PBT-Lap	6	12	63	65
BT-RuleVI	6	17	138	31
PBT-RuleVI	6	12	38	29
Rulex	6	17	59	16

Supporting facts: *output1_pred5((body_octagon, jacket_yellow),*
hidden1_pred3(head_round, body_octagon, jacket_yellow).

Data base: Monk2 using GYAN (PCC-*LAP*) (Table 6.38)

Query: *?monk2(head_round, body_round, is_smiling, holding_sword, jacketnt_red, hasnt_tie)*

Response: *false*

Supporting facts: *output1_pred2(head_round, jacketnt_red, hasnt_tie),*
hidden1_pred9(body_round, is_smiling, holding_sword).

Data base: Monk2 using GYAN (PCC-*RuleVI*) (Table 6.38)

Query: *?monk2(head_round, bodynt_round, is_smiling, holdingnt_sword, jacketnt_red, X)*

Response: *true*

Supporting facts: *monk2_pred5(head_round, bodynt_round, is_smiling, holdingnt_sword, jacketnt_red, hasnt_tie),*

Data base: Monk3 using GYAN (PCC-*RuleVI*) (Table 6.39)

Query: *?monk3(body_round, holding_sword, jacket_red)*

Response: *true*

Table 6.39: SHRUTI knowledge base for the Monk3 data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-Lap	6	17	157	45
PCC-Lap	3	9	23	11
CC-RuleVI	6	17	40	19
PCC-RuleVI	3	9	12	6
BT-Lap	6	17	258	70
PBT-Lap	3	10	13	6
BT-RuleVI	6	17	59	10
PBT-RuleVI	3	10	14	7
Rulex	6	17	8	3

Table 6.40: SHRUTI knowledge base for the Remote Sensing data set recognizing water and forest

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-Lap	3	15	130	14
PCC-Lap	3	15	42	8
CC-RuleVI	3	15	93	13
PCC-RuleVI	3	15	12	6
BT-Lap	3	15	135	14
PBT-Lap	3	15	42	8
BT-RuleVI	3	15	81	12
PBT-RuleVI	3	15	12	8
Rulex	3	15	44	4

Supporting facts: *monk3_pred0(body_round, jacket_red),*

Data base: Monk3 using GYAN (BpTower-LAP) (Table 6.39)

Query: *?monk3(body_octagon, holding_sword, jacket_blue)*

Response: *false*

Supporting facts: *output_pred6(body_octagon, jacket_blue),*
hidden1_pred0(jacket_blue), hidden1_pred0(body_octagon),
hidden1_pred4(body_octagon, holding_sword).

Data base: Remote Sensing using GYAN (PCC-LAP) (Table 6.40)

Query: *?rs(red_medium, green_low, blue_high)*

Table 6.41: SHRUTI knowledge base for the Mushroom data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
PCC-RuleVI	11	29	24	12
PCC-Lap (FD)	6	25	608	19
CC-RuleVI (FD)	6	45	92	9
PCC-RuleVI(FD)	6	24	24	8
PBT-RuleVI	10	25	26	18
PBT-Lap (FD)	6	19	128	29
BT-RuleVI (FD)	6	45	124	11
PBT-RuleVI (FD)	6	19	11	7
Rulex	19	95	125	4

Response: *true*

Supporting facts: *water0(red_medium, green_low, blue_high),*

Data base: Remote Sensing using GYAN (*RULEX*) (Table 6.40)

Query: *?rs(red_low, green_medium, blue_high)*

Response: *true*

Supporting facts: *forest0(red_low, green_medium, blue_high)*

Data base: Mushroom using GYAN (PCC-*RuleVI*) (Table 6.41)

Query: *?edible(odor_almond, gill-size_broad, gill-color_not-buff, stalk-surface-above-ring_not-scaly, stalk-surface-below-ring_not-scaly, stalk-color-above-ring_orange, stalk-color-below-ring_orange, ring-type_flaring, spore-color_brown, population_not-clustered, habitat_waste)*

Response: *true*

Supporting facts: *edible1(odor_almond, stalk-surface-above-ring_not-scaly, stalk-surface-below-ring_not-scaly, spore-color_brown, population_not-clustered)*

Data base: Voting using GYAN (PBT-*RuleVI*) (Table 6.42)

Query: *?democrat(water_sharing, infant_handicap, adapt-budget-resolution_no, education_spending, physician-fee_no, elsavador-aid_no, religious-group-in-schools_yes,*

Table 6.42: SHRUTI knowledge base for the Voting data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
PCC-Lap	11	24	334	121
CC-RuleVI	16	48	79	79
PCC-RuleVI	11	24	30	11
PBT-RuleVI	13	28	65	43
Rulex	16	48	13	5

Table 6.43: SHRUTI knowledge base for the Moral Reasoner data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-RuleVI	19	40	43	36
PCC-RuleVI	12	26	49	39
BT-RuleVI	23	48	116	80
PBT-RuleVI	15	30	62	36
Rulex	8	16	4	5

missile-mx_yes, immigration_no, crime_no, duty-free-export_not-sure, administration-south-africa_no)

Response: *true*

Supporting facts: *democrat5(infant_handicap, elsavador-aid_no, religious-group-in-schools_yes, missile-mx_yes, immigration_no, crime_no),*

democrat14(physician-fee_no, adapt-budget-resolution_no, admin-south-africa_no)

Data base: Moral reasoner using GYAN (cascade-*RuleVI*) (Table 6.43)

Query: *?guilty(plan-known_yes, someone-else-cause-harm_no, out-rank-perpetrator_no, monitor_yes, harm-caused-as-planned_yes, goal-outweigh-harm_yes, foresee-inter_yes, external-cause_no, control-perpetrator_yes, benefit-protagonist_yes, careful_yes, benefit-victim_no, severity-harm_yes, achieve-goal_no, intervening-contribution_no, foresee-ability_yes, external-force_no, mental-state_intended, necessary-for-harm_yes)*

Response: *true*

Supporting facts: *guilty_pred0(benefit-victim_no, severity-harm_yes, intervening-*

Table 6.44: SHRUTI knowledge base for the Cleveland heart disease data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-RuleVI	13	38	161	122
PCC-RuleVI	11	34	136	99
BT-RuleVI	13	38	185	134
PBT-RuleVI	11	34	159	111
Rulex	12	43	130	5

contribution_no, external-force_no),

guilty_pred1(foresee-intervention_yes, benefit-victim_no, severity-harm_yes, external-force_no, mental-state_intended)

Data base: Moral reasoner using GYAN (BpTower-*RuleVI*) (Table 6.43)
Query: *?guilty(sufficient-for-harm_yes, produce-harm_yes, plan-known_yes, plan-include-harm_yes, someone-else-cause-harm_no, out-rank-perpetrator_no, monitor_yes, harm-caused-as-planned_yes, goal-outweigh-harm_yes,goal-achieve-able-less-harmful_yes, foresee-intervention_yes, external-cause_no, control-perpetrator_yes, benefit-protagonist_yes, careful_yes, benefit-victim_no, severity-harm_yes, achieve-goal_no, intervening-contribution_no, foresee-ability_yes, external-force_no, mental-state_negligent, necessary-for-harm_yes)*
Response: *Do not have enough information*

Data base: Cleveland using GYAN (PCC-*RuleVI*) (Table 6.44)
Query: *?healthy-patient(age_41-50, chest-pain_asympt, resting-bp_109-138, cholesterol_126-229, resting-ecg_normal, max-heart-rate_71-142, exer-induced-anigna_no, old-peak_0-2.1, slope_up, vessel-color_color1, thal_normal)*
Response: *true*
Supporting facts: *healthy-patient-pred89(age_41-50, chest-pain_asympt, resting-bp_109-138, resting-ecg_normal, exer-induced-anigna_no, old-peak_0-2.1, slope_up, vessel-color_color1, thal_normal)*

healthy-patient-pred81(resting-bp_109-138, cholesterol_126-229,

Table 6.45: SHRUTI knowledge base for the Breast Cancer data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
CC-RuleVI	9	90	340	288
PCC-RuleVI	9	42	46	37
BT-RuleVI	9	90	274	192
PBT-RuleVI	9	42	60	45
Rulex	9	36	200	10

Table 6.46: SHRUTI knowledge base for the Queensland Rail data set

Gyan using	No of concepts	No of sub-concepts	No of facts	No of predicates
PCC-Lap	7	27	31	12
CC-RuleVI	14	58	97	19
PCC-RuleVI	8	29	12	7
Rulex	9	52	150	2

max-heart-rate_71-142, exer-induced-anigna_no, old-peak_0-2.1, slope_up, thal_norm)

Data base: Breast cancer using GYAN (PBT-*RuleVI*) (Table 6.45)

Query: *?benign(clump-thickness_10, cell-size_10, cell-shape_5, margin_4, epithelial-cell-size_10, nuclei_9, chromation_8, nucleoli_8, mitosis_10)*

Response: *false*

Supporting facts: *benign-pred14(clump-thickness_10, cell-size_10, margin_4, epithelial-cell-size_10, nucleoli_8, mitosis_10)*

benign-pred35(cell-shape_5), benign-pred42(nuclei_9)

Data base: Queensland Rail using GYAN (PCC-*RuleVI*) (Table 6.46)

Query: *?risky_track(type_public, protect_nil, track_1or2, surface_concrete, speed_fast, road-visible_poor, pedestr_medium, approach-signs_yes)*

Response: *true*

Supporting facts: *risky_track_pred0(road-visible_poor)*

risky_track_pred1(speed_fast, pedestr_medium)

risky_track_pred2(surface_concrete)

6.4.2 Phase4: Summary and discussion

The GYAN methodology provides a basis for forming restricted first-order semantic networks from the knowledge embedded in ANNs, that can be further interfaced with a knowledge base reasoner to perform inferences. The rules are represented by links between predicates or networks of predicates. The links between the nodes within a predicate node and possibly to a number of fact nodes are able to represent true instantiation of predicates *i.e.* facts. The motivation for providing such a semantic network is to capture the ease with which humans efficiently draw on a wide range of inferences given some simple stimulus. These automated semantic networks are able to perform inferences when realized on SHRUTI reasoning system.

When the SHRUTI network is built according to the automated knowledge-base generated utilizing the *LAP* algorithm in GYAN phase3, the semantic depth of a query corresponds to the number of hidden nodes in the trained cascade and BpTower networks. For knowledge-bases generated by component algorithms other than *LAP*, the semantic depth is always one. To support ANN architectures, the generated type-hierarchy is of single depth, according to the attributes and the values they contain.

The need to fulfill SHRUTI's restriction that all the variables in antecedent predicates of a rule must appear in the consequent predicate of the rule, means that the target predicate in automated knowledge bases (corresponding to the output node of a trained ANN) usually contains a large number of arguments. SHRUTI has a restriction (upper bound) on the number of distinct phases for representing all the entities that must remain active at any given time in order to avoid cross-talk etc. The SHRUTI system entirely depends on the scope of phase distribution during an episode of reasoning; that is, each entity is assigned a distinct phase while encoding of rules. According to Shastri

and Ajjanagadde [1993], the number of distinct entities that can occur as argument fillers in the dynamic facts representation cannot exceed $\mid \pi_{max}/w \mid$, where π_{max} is the maximum period at which an argument node can sustain synchronous oscillations, and w equals the width of the window of synchrony. Further, the authors determine the number of entities referenced by the dynamic facts to be 7 ± 2.

The SHRUTI implementation [Hayward, 1999] used for the testing reported in this thesis is capable of responding efficiently to queries involving more arguments than the suggested 7 ± 2 [Shastri and Ajjanagadde, 1993]. The reason is the way the synchronous oscillations between nodes are realized in the implementation of the SHRUTI model. The data structure that is used to represent the behavior of a node over a set period is simply an unsigned integer, a set of bits, similar to the implementation by Mani and Shastri [1995]. As explained above, the behavior of nodes in the SHRUTI model is subjected to their receiving or not receiving inputs over a set of period of time, π. The firing of an argument node is represented by a single bit of the integer being set. The use of integers to represent the temporal sequence degrades the oscillatory behavior of the SHRUTI system, and the inference capability is also degraded correspondingly.

The query performance of GYAN phase4 is as follows:

- The responses to the posed queries reflect the quality of the automated knowledge bases (in other words the ANN solutions and the extracted rules) for the given problem domain.

- The time taken to answer a query ranges from a fraction of a second to a few tens of seconds, corresponding to the size of the knowledge base (number of fired rules, facts and arguments involved in the query).

- The reasoning has been shown to work well (*i.e.* the system returns the expected answers) when attributes are left unbound (equivalent to ask-

ing: 'is this true for any values of x?'). In this particular implementation of SHRUTI [Hayward, 1999], queries consisting of more than one variable are not guaranteed to respond correctly. In principle, SHRUTI is capable of answering queries with multiple variables by assigning a unique phase to each distinct variable along with the constants.

- Examination of the results shows that queries of semantic depth 1 are responded to correctly until the number of arguments exceeds 19. Queries of depth 3 are responded to correctly until 9 arguments are involved. The knowledge base generated from the BpTower network utilizing *Rule VI* for the Moral Reasoner problem solution, which consists of a target predicate with 23 arguments (Table 6.43) did not respond correctly to queries. Similarly the knowledge base generated from the cascade network utilizing *LAP* for the Cleveland heart disease problem, which consists of a target predicate with 11 arguments (Table 6.44) incorrectly responds to the queries posed.

- Due to the relative flatness in inferential depth of input space in ANNs, much more stringent queries are required in order to test the quality of automated knowledge bases in the SHRUTI model. The queries are not obtained at the cost of certain loss of fidelity, as there will not be a type-hierarchy link larger than steps in the query, and all the supporting facts will easily be matched. Sometimes, during a reasoning episode the reasoner does not find an answer within a reasonable time, as the reasoner is unable to activate the facts (that are present in the knowledge base and is related to the query) precisely at the time of reasoning (they may get activated late but then they are not matched due to progression of the reasoning cycle).

- In general, the automated knowledge bases utilizing the *LAP* decompositional rule-extraction technique are larger than those obtained from other component programs. The reason is that the actual structure of the trained ANN is preserved in the automated knowledge base, a predicate

for each non-input node in the ANN.

This type of knowledge base is important when the inference engine is asked to classify unknown instances in the form of queries, and the response is given based on rules and facts summarizing the data sets. The trained ANN is only capable of classifying if the instance is a member of the target class or not. The interfacing of the knowledge embedded in the trained ANN with an inference engine facilitates an explanation of why this decision has been made. This is especially important for partially instantiated cases. The inference engine produces a response with all possible bindings to unspecified attributes. The explanation of the query includes concepts, predicates and rules activated during the inference process. The rapid response to a query posed to GYAN using the SHRUTI knowledge base system is particularly useful for an on-line system using and understanding the trained ANN solutions of the given problem domain.

6.5 Relative performance of Gyan

The results show several relationships between the different data sets and overall accuracy of the inductive algorithms such as GYAN, FOIL and C4.5. Some of the observations based on the datasets used in this thesis are:

- The experiments confirm the previous results[2] that a parallel task (such as Monk2, Remote-sensing, etc.) is learnt better by GYAN (a connectionist method) than by FOIL or C4.5 (symbolic methods). On the other hand, a sequential task (such as Mushroom, Cleveland, Moral Reasoner etc.) is learnt better by FOIL or C4.5 as compared to GYAN.

- In general, the generalization accuracy (when moving from training to test data) of FOIL is worse than GYAN. The generalization accuracy

[2]The previous researchers [Quinlan, 1993b] have pointed out that connectionist methods work better for *parallel tasks* (where all the input variables are relevant to the classification) and symbolic methods are more suitable for *sequential tasks* (where the relevance of a particular input variable depends on the values of other input variables).

even becomes worse when the dataset has noise (such as Monk3, QR).

- GYAN gives more accurate results as compared to FOIL and C4.5 when noise (imperfection) is present in data (such as Monk3). In such situations, GYAN takes the advantage of distributed codings.

- GYAN performed (in terms of accuracy and comprehensibility) better than FOIL and C4.5 when relatively small amount of data (such as Remote-sensing) is available for training. When a large number of data (such as Mushroom, Cleveland, etc.) is available for training, FOIL and C4.5 performed better than GYAN.

- Datasets in which the distribution of patterns (belonging to positive or negative classes) is quite uneven, there is not much difference between the accuracy performance of GYAN and C4.5. But the performance of FOIL is worse than these two.

Based on these results it can be said that GYAN, combining the expressiveness of symbolic representations with the adaptiveness of connectionist systems, is capable of providing a good predictive accuracy and good expressiveness of (output) language depending upon the nature of datasets.

6.6 Chapter summary

This chapter presents a series of experiments that empirically evaluates GYAN in the context of classification learning tasks. The purpose of the experiments is to evaluate the effectiveness of GYAN along the dimensions of explanatory power, accuracy, fidelity and comprehensibility. The first section in this chapter evaluates GYAN phase1, with three incremental neural architectures each of which starts with only an input layer and an output layer (and a single hidden node in the case of CEBPN) and adapts the architecture to the learning task by sequentially inserting hidden nodes. The second section evaluates the pruning algorithm to eliminate redundant nodes and links in the trained ANNs. The FD algorithm is evaluated for comparison of results. The third

section evaluates the predicate rules generated from the ANNs. The generated rule-sets are compared with those of FOIL and C4.5. The last section evaluates the restricted first-order semantic networks generated for the problem domains which are interfaced with SHRUTI to allow inferencing. Thus, the proposed system GYAN allows interacting with a data set through queries.

For each of the data sets, the results obtained using GYAN compare favorably with those reported by other authors and those generated with other algorithms (FOIL and C4.5). The successful application and competitive results obtained by GYAN for various problem domains demonstrate its effectiveness in real-life problems (such as Queensland Rail and Remote Sensing), in fairly large size problems (in terms of number of attributes, such as Breast Cancer, Moral Reasoner, Voting and Mushroom), and in continuous-valued problem domains (such as Cleveland heart disease). Importantly, the agreement in the results obtained by different components at any stage of GYAN demonstrates the efficiency of the framework. The development and success of GYAN (and also of LINUS and FOIL) shows that propositional rule-extraction techniques can be effectively extended to represent knowledge embedded in trained ANNs in the form of first-order logic language with some restrictions. This success also emphasizes the importance of reliable and general propositional learners.

Chapter 7

Conclusion

Artificial neural network learning methods provide a robust and non-linear approach to approximating the target function for many classification, regression and clustering problems. The ability to learn and generalize from data, which mimics the human capability of learning from experience, makes ANNs a good alternative to competitive symbolic methods for inductive learning tasks. Despite a demonstrated good predictive performance in a wide variety of practical problems, there have always been researchers unsatisfied with the general representation of knowledge in ANNs. A fundamental consideration behind this shortcoming is the poor comprehensibility of the learned model, and the inability to generate and represent explanation structures such as reasoning paths, expectation failure, etc.

This thesis addresses the knowledge representation issue by developing procedures to explain the decision process in neural networks in the form of symbolic rules (predicate rules with variables), and inserting the generic and conceptual knowledge embedded in ANNs into a connectionist rule-based reasoning system to allow user interface and explanation capability.

A multi-stage methodology GYAN is developed and evaluated for the task of extracting knowledge from the trained ANNs. The extracted knowledge is represented in the form of restricted first-order logic rules, and subsequently

allows user interaction by interfacing with a knowledge based reasoner. The performance of GYAN is demonstrated using a number of real world and artificial data sets combining pattern recognition and decision making problems. The empirical results demonstrate that: (1) an equivalent symbolic interpretation is derived describing the overall behavior of the ANN with high accuracy and fidelity, and (2) a concise explanation is given (in terms of rules, facts and predicates activated in a reasoning episode) as to why a particular instance is being classified into a certain category.

The concluding chapter of this thesis discusses the contributions of the presented research and proposes several future extensions to overcome the limitations of this research work.

7.1 Contributions

In particular, the major contributions of this thesis include:

- the development of the general, domain-independent, multi-stage, hybrid GYAN methodology to combine the expressiveness and procedural versatility of symbolic systems with the fuzziness and adaptiveness of connectionist representations, such that the well-established ideas of propositional techniques are utilized to represent the knowledge embedded in ANNs in a constrained first-order language. The integration of various components in GYAN allows it to apply to a variety of feedforward neural network architectures such as cascade type of networks and multi-layer architectures using constrained backpropagation learning.

- extensive empirical evaluation of each stage in GYAN. The methodology compares favorably with those reported by other authors and with other algorithms (FD, FOIL and C4.5). The successful application and competitive results obtained by GYAN for various problem domains (real-life, fairly large size and continuous-valued) demonstrate its effectiveness and overall support for the idea of using a combination of neural networks

and rule-extraction techniques. Importantly, the consistency in the results obtained by various components at any stage of GYAN demonstrates the efficiency of the framework.

- an extensive survey of techniques of rule extraction from neural networks published to date, an empirical evaluation of various rule-extraction techniques (pedagogical vs decompositional) to compare the principles of each and also a brief discussion of some of the leading symbolic techniques including propositional and ILP systems.

- the provision of a concise explanation and inferencing capability by interfacing the knowledge embedded in ANNs with knowledge based reasoners, instead of merely providing an equivalent symbolic interpretation describing the overall behavior of ANNs. The incorporation of several rule-extraction and neural learning algorithms makes the construction of knowledge bases feasible to meet various requirements such as accuracy, fidelity and comprehensibility.

- a detailed discussion of how Plotkin's *least general generalization of literals* concept can be extended to map propositional to predicate rules with variables. The concept description becomes more expressive (with the appearance of predicates and variables in rules) and comprehensible (with reduction in the number of rules) by applying the generalization algorithm. The extension of expressiveness in symbolic rules (from propositional to predicate) avoids the serious degradation of generalization performance (decreasing classification accuracy when moving from examples to test data) that occurs in learning of predicate (first-order) rules directly from data.

- the development of the pruning algorithm (inspired by the *MofN* algorithm [Towell and Shavlik, 1993]) for the identification of relevant attributes/values in the trained ANNs as a pre-cursor to rule-extraction techniques. A number of experiments are conducted to demonstrate the benefit of reducing the input space after neural network training on the

complete data set (relative to reducing the input space before neural learning). Experiments are also conducted to show that projection to a space of lower dimensionality translates a neural network into a compact rule-set that seems impossible to solve otherwise.

7.2 Future extensions

A number of limitations of the GYAN methodology have been identified during its development that reduce its usefulness in terms of functionality and robustness. The primary limitation of GYAN is the absence of recursively defined predicates and infinite terms in presentation of rules. Researchers have shown that connectionist systems cannot successfully represent recursive symbolic structures in their architectures [Kalinke, 1997]. This necessitates the introduction of recursive terms at a later time, when the propositional rules extracted from ANNs are mapped to predicate rules. One approach in this direction could be the utilization of extracted propositional rules (knowledge learned by an ANN) as background knowledge, with first-order rules learned by applying some bottom-up (such as Plotkin's lgg) or top-down (such as FOIL or MIS) processing when presented with examples.

The knowledge base generated by the GYAN methodology can be utilized widely if interfaced to a Prolog inference engine. The Prolog knowledge base can be generated by making a few changes in the knowledge base parser, mostly to eliminate the constraints required by the SHRUTI knowledge base reasoner.

Despite the fact that recursively defined predicates and infinite terms are not allowed in the generated knowledge representation, the formalism is appropriate for real-life problems. The development and success of GYAN (and also that of LINUS and FOIL) has shown that propositional rule-extraction techniques can be effectively extended to represent knowledge embedded in the trained ANN in the form of first-order language with some restrictions,

which additionally highlight the importance of reliable and general propositional learners. One solution to this task will be the direct utilization of clustered weight states (that are fewer in numbers) produced after pruning the ANN for rule generation. The weight elements in a cluster may not show any dependencies (as seen in the empirical evaluation in chapter 6), but the relevant attributes are recognized during the pruning process. The application of some careful weight vector analysis will directly output the dependencies as rules and will reduce the complexity of any subsequent rule-extraction technique.

Another approach is to improve the existing rule-extraction techniques in GYAN. The distinguishing feature of *LAP* is high fidelity of the extracted rules because of the exhaustive search in weight space. For large problem domains, this type of search-based method fails to extract symbolic rules from the learned weights. A sequential covering approach (such as in the AQ or CN2 symbolic learning methods) should be utilized to apply *LAP* in large problem domains. Instead of checking the whole space, the rules should be extracted according to their predictive accuracy on training patterns. Each rule should be checked to see how many patterns are covered by this rule. The covered patterns should be removed from the set of examples. The further extraction should be continued according to the accuracy and comprehensibility requirements. The process can be terminated when a fair number of patterns are covered by the extracted rule-set.

A similar set-covering approach could be applied in *RuleVI* in which each pattern is eliminated sequentially after the rules are generated for that pattern. Due to the consideration of each pattern in the data-set separately, a large number of rules are generated, sometimes more than the number of examples (as each example is multiply checked for each sparse-coded input dimension). The size of the rule-set can be reduced by checking the accuracy of the rule, generated by an example instance, on other instances in the same

example set. The examples that are covered by this rule should be removed from the pattern set instead of considering them for the generation of rules as in the original version. Also if the accuracy of the extracted rule is very low (some pre-defined bound) on other example instances then this rule can be dropped from the rule-set.

The current GYAN methodology includes the reduction of redundant conjunctive expressions extracted by a rule-extraction method as a part of the generalization algorithm for the mapping of propositional to predicate rules. The independent implementation of the reduction procedure will make it widely applicable to any propositional technique for reducing the extracted DNF expression.

One of the directions to take in future is the parallel mapping of a connectionist knowledge base reasoner (such as SHRUTI) onto a network of workstations or other parallel machines to solve large and complex problems.

7.3 Final remarks

The idea of extracting symbolic rules from ANNs to provide a decision mechanism has been quite popular among researchers in recent years. The newly found interest in applying neural networks for the *knowledge discovery and data mining* tasks has also provided much impetus to the process of understanding the internal mechanism of trained neural networks. The representation of knowledge in first-order language (with some limitations) opens the possibility of gaining comprehensible rules from ANNs. The insertion of learned rules from ANNs into knowledge based reasoners allows the provision of a user interface. By this means the GYAN methodology increases the potential of applying ANNs to various *knowledge discovery and data mining tasks* and to the *automation of knowledge acquisition tasks.*

Appendix A

The generated rule sets

Some of the predicate rules generated by GYAN for all the problem domains studied in this thesis are reported in this appendix.

A.1 The Monk1 problem domain

- Target predicate: monk1(head_shape, body_shape, jacket_color)

  - Rules extracted using PCC-LAP

    monk1(X,Y,Z) $\Leftarrow$ output1_pred0(Z) $\wedge$ $\neg$hidden1(X,Y,Z)
    $\neg$hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred1(Z)
    $\neg$hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred3(X,X)
    output1_pred0(red), hidden1_pred1(red)
    hidden1_pred3(round, round)

    monk1(X,Y,Z) $\Leftarrow$ output1_pred2(X) $\wedge$ $\neg$hidden1(X,Y,Z)
    $\neg$hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred2(Y)
    output1_pred2(square), hidden1_pred2(square)

    monk1(X,Y,Z) $\Leftarrow$ output1_pred1(Y) $\wedge$ $\neg$hidden1(X,Y,Z)
    $\neg$hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred0(X)
    output1_pred1(octagon), hidden1_pred0(octagon)

  - Rules extracted using PCC-RuleVI

    monk1(X,X,Z) $\Leftarrow$ monk1_pred0(X,X)
    monk1(X,X,Z) $\Leftarrow$ monk1_pred1(X,Y)
    monk1(X,Y,Z) $\Leftarrow$ monk1_pred2(Z)
    monk1_pred0(round,round), monk1_pred0(square,square),
    monk1_pred0(octagon,octagon)
    monk1_pred1(round,square), monk1_pred1(square,octagon)
    monk1_pred2(red)

- Rules extracted using PBT-LAP

monk1(X,Y,Z) $\Leftarrow$ output1_pred0(Z) $\wedge$ hidden1(X,Y,Z)
hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred0(Z)
hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred1(X,X)
output1_pred0(red), hidden1_pred0(red)
hidden1_pred1(square,square), hidden1_pred1(octagon,octagon)

monk1(X,Y,Z) $\Leftarrow$ output1_pred1(X) $\wedge$ hidden1(X,Y,Z)
hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred2(Y)
output1_pred1(round), hidden1_pred2(round)

$\neg$monk1(X,Y,Z) $\Leftarrow$ output1_pred5(Y,Z) $\wedge$ hidden1(X,Y,Z)
$\neg$hidden1(X,Y,Z) $\Leftarrow$ hidden1_pred3(X,Y,Z)
output1_pred5(octagon,yellow), hidden1_pred3(round,octagon,yellow)

- Rules generated using C4.5

monk1(X,X,Z) $\Leftarrow$ monk1_pred0(Z)
monk1(X,Y,Z) $\Leftarrow$ monk1_pred1(X,X)
monk1_pred0(red)
monk1_pred1(round,round), monk1_pred1(square,square),
monk1_pred1(octagon,octagon)

- Rules generated by Foil

Target predicate: monk1(Head_shape, Body_shape, Is_smiling,
holding, Jacket_color, Has_tie)

is_monk1(X,X,Z,U,V,W)
is_monk1(X,Y,Z,U,red,W)

A.2 The Monk2 problem domain

- Target predicate: monk1(head_square, body_square, is_smiling, holding, jacket_color, has_tie)

  - Rules extracted using PCC-LAP

    monk2(X,Y,Z,U,V,W) $\Leftarrow$ output1_pred0(V,W) $\wedge$ $\neg$hidden1(X,Y,Z,U,V,W)
    $\neg$hidden1(X,Y,Z,U,V,W) $\Leftarrow$ hidden1_pred20(X,Y,Z,U)
    output1_pred0(red,yes), hidden1_pred20(nt-round,nt-round,no,nt-sword)

    $\neg$monk2(X,Y,Z,U,V,W) $\Leftarrow$ output1_pred2(X,Z,W) $\wedge$ hidden1(X,Y,Z,U,V,W)
    $\neg$hidden1(X,Y,Z,U,V,W) $\Leftarrow$ hidden1_pred9(Y,U,V)
    output1_pred2(round,yes,no), hidden1_pred9(round,sword,nt-red)

  - Rules extracted using PBT-LAP

    monk2(X,Y,Z,U,V,W) $\Leftarrow$ output1_pred1(U,W) $\wedge$ hidden1(X,Y,Z,U,V,W)
    hidden1(X,Y,Z,U,V,W) $\Leftarrow$ hidden1_pred2(X,Y,Z,V)
    output1_pred1(sword,yes), hidden1_pred2(nt-round,nt-round,no,nt-red)

  - Rules extracted using PCC-RuleVI

    monk2(X,Y,Z,U,V,W) $\Leftarrow$ monk2_pred5(X,Y,Z,U,V,W)
    monk2_pred5(round,nt_round,yes,nt_sword,nt_red,no)

    monk2(X,Y,Z,U,V,W) $\Leftarrow$ monk2_pred13(X,Y,Z,U,V)
    monk2_pred13(nt_round,nt_round,no,nt_sword,nt_red}

- Rules extracted using Foil

  is_monk2(X,square,yes,U,yellow,W).

  is_monk2(square,octagon,yes,flag,V,W).

  is_monk2(X,square,square,U,green,No).

A.3 The Monk3 problem domain

- Target predicate: monk3(body_square, holding, jacket_red)

  - Rules extracted using PCC-RuleVI

    monk3(X,Y,Z) $\Leftarrow$ monk3_pred0(X, Z)
    monk3_pred0(round,red), monk3_pred0(round,green),
    monk3_pred0(round,yellow), monk3_pred0(square,yellow),
    monk3_pred0(square,red), monk3_pred0(square,green)

    $\neg$monk3(X,Y,Z) $\Leftarrow$ monk3_pred1(X, Z)
    monk3_pred1(octagon,blue)

    $\neg$monk3(X,Y,Z) $\Leftarrow$ monk3_pred2(X, Y)
    monk3_pred2(octagon,nt-sword)

    monk3(X,Y,Z) $\Leftarrow$ monk3_pred3(Y, Z)
    monk3_pred3(sword,green)

  - Rules extracted using PBT-LAP

    monk3(X,Y,Z) $\Leftarrow$ output1_pred0(Y, Z)
    output1_pred0(sword,green)

    monk3(X,Y,Z) $\Leftarrow$ output1_pred1(X, Z)
    output1_pred1(round,red), output1_pred1(round,green),
    output1_pred1(round,yellow), output1_pred1(square,yellow)
    output1_pred1(square,red), output1_pred1(square,green),

¬monk3(X,Y,Z) ⇐ output1_pred2(X, Z)

output1_pred2(octagon,blue)

¬monk3(X,Y,Z) ⇐ output1_pred3(X, Y)

output1_pred3(octagon,balloon), output1_pred3(octagon,flag)

- Rules extracted using Foil

is_monk3(X,square,Z,sword,green,W).

is_monk3(X,round,Z,U,yellow,W).

is_monk3(X,round,Z,U,red,W).

A.4 The Remote sensing problem domain

- Target predicate: rs(red, green, blue)

  - Rules extracted using PCC-LAP

    rs(X,Y,Z) $\Leftarrow$ water0(X,Y,Z)
    water0(very_low,low,high), water0(very_low,low,very_high),
    water0(very_low,medium,high), water0(high,medium,very_high),
    water0(low,low,high), water0(medium,very_low,high),
    water0(low,medium,high)

    $\neg$rs(X,Y,Z) $\Leftarrow$ forest3(Y)
    forest3(high), forest3(very_high)

  - Rules extracted using RULEX

    rs(X,Y,Z) $\Leftarrow$ water0(X,Y,Z)
    water0(very_low,medium,high), water0(low,high,very_high),
    water0(very_low,high,very_high), water0(low, medium, high)

    rs(X,Y,Z) $\Leftarrow$ forest0(X,Y,Z)
    forest0(low,very_low,very_low), forest0(low,low,low),
    forest0(high,very_low,medium), forest0(medium,low,medium)
    forest0(medium, very_low, very_low)

    $\neg$rs(X,Y,Z) $\Leftarrow$ forest1(X,Y,Z)
    forest1(high,low,very_low)

A.5 The Mushroom problem domain

- Rules extracted using PCC-RuleVI (FD)

target predicate: edible(cap-color, odor, stalk-surface-above-ring, spore-color, population, habitat)

edible(A,B,C,D,E,F) $\Leftarrow$ edible_pred0(B)
edible_pred0(almond), edible_pred0(none)

edible(A,B,C,D,E,F) $\Leftarrow$ edible_pred1(B,D,E)
edible_pred1(anise,brown,solitary)

$\neg$edible(A,B,C,D,E,F) $\Leftarrow$ edible_pred2(B)
edible_pred2(foul), edible_pred2(spicy), edible_pred2(fishy)

$\neg$edible(A,B,C,D,E,F) $\Leftarrow$ edible_pred3(B,E)
edible_pred3(pungent,clustered)

$\neg$edible(A,B,C,D,E,F) $\Leftarrow$ edible_pred4(B,C)
edible_pred4(pungent,scaly)

$\neg$edible(A,B,C,D,E,F) $\Leftarrow$ edible_pred5(C,E)
edible_pred5(scaly,clustered)

- Rules extracted using PCC-RuleVI

target predicate: edible(odor, gill-size, gill-color, stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring, stalk-color-below-ring, ring-type, spore-color, population, habitat)

edible(A,B,C,D,E,F,G,H,I,J,K) $\Leftarrow$ edible_pred1(A,D,E,I,J)

edible_pred1(almond,not-scaly,not-scaly,brown,not-clustered)

$\neg$edible(A,B,C,D,E,F,G,H,I,J,K) $\Leftarrow$ edible_pred5(A,B,G,I,K)

edible_pred5(pungent,narrow,nt-orange,brown,not-waste)

$\neg$edible(A,B,C,D,E,F,G,H,I,J,K) $\Leftarrow$ edible_pred0(A,F,G,H,I,K)

edible_pred0(pungent,nt-orange,nt-orange,large,black,nt-waste)

A.6 The Voting problem domain

• Target predicate: democrat(water-project, adapt-budget-resolution, physician-fee, elsavador-aid, missile-mx, immigration, corporation-cutback, superfund-sue, crime, duty-free-export, administration-south-africa)

• Rules extracted using PCC-RuleVI

democrat(A,B,C,D,E,F,G,H,I,J) $\Leftarrow$ democrat_pred3(C,G)

democrat_pred3(no,not-sure)

democrat(A,B,C,D,E,F,G,H,I,J) $\Leftarrow$ democrat_pred8(C,E,F)

democrat_pred8(no,yes,no)

democrat(A,B,C,D,E,F,G,H,I,J) $\Leftarrow$ democrat_pred9(B,C)

democrat_pred9(yes,no)

democrat(A,B,C,D,E,F,G,H,I,J) $\Leftarrow$ democrat_pred10(E,F,G,H,K)

democrat_pred10(yes,no,not-sure,yes,no)

$\neg$democrat(A,B,C,D,E,F,G,H,I,J) $\Leftarrow$ democrat_pred1(C,E,F)

democrat_pred1(no,not-sure,no)

A.7 The Moral Reasoning problem domain

• Rules extracted using cascade-RuleVI

target predicate: guilty(plan-known, someone-else-cause-harm, out-rank-perpetrator, monitor, harm-caused-as-planned, goal-outweigh-harm, foresee-intervention, external-cause, control-perpetrator, benefit-protagonist, careful, benefit-victim, severity-harm, achieve-goal, intervening-contribution, foresee-ability, external-force, mental-state, necessary-for-harm)

guilty(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S) $\Leftarrow$ guilty_pred0(L,M,O,Q)
guilty_pred0(no,yes,no,no)

guilty(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S) $\Leftarrow$ guilty_pred1(G,L,M,Q,R)
guilty_pred1(yes,no,yes,no,intended)

• Rules extracted using PCC-RuleVI

target predicate: guilty(plan-known, out-rank-perpetrator, control-perpetrator, benefit-protagonist, careful, benefit-victim, severity-harm, achieve-goal, intervening-contribution, external-force, mental-state, necessary-for-harm)

guilty(A,B,C,D,E,F,G,H,I,J,K,L) $\Leftarrow$ guilty_pred6(A,E,F,J,K)
guilty_pred6(yes,no,no,no,intended)

$\neg$guilty(A,B,C,D,E,F,G,H,I,J,K,L) $\Leftarrow$ guilty_pred13(F,J,K)
guilty_pred13(yes,yes,negligent)

$\neg$guilty(A,B,C,D,E,F,G,H,I,J,K,L) $\Leftarrow$ guilty_pred19(G,J)
guilty_pred19(no,yes)

A.8 The Cleveland heart disease problem domain

• Target predicate: healthy(age, chest-pain, resting-bp, cholesterol, resting-ecg, max-heart-rate, exer-induced-anigna, old-peak, slope, vessel-color, thal)

• Rules extracted using PCC-RuleVI

¬healthy(A,B,C,D,E,F,G,H,I,J,K) ⇐ healthy_pred2(A,B,C,E,F,I,J,K)
healthy_pred2(51-68,abnang,109-138,hyper,143-163,flat,color1,normal)

¬healthy(A,B,C,D,E,F,G,H,I,J,K) ⇐ healthy_pred8(B,C,E,F,I,J,K)
healthy_pred8(asympt,109-138,hyper,71-142,flat,color1,rever)

healthy(A,B,C,D,E,F,G,H,I,J,K) ⇐ healthy_pred81(C,D,F,G,H,I,K)
healthy_pred81(109-138,126-229,71-142,no,0-2.1,up,normal)

healthy(A,B,C,D,E,F,G,H,I,J,K) ⇐ healthy_pred89(A,B,C,E,G,H,I,J,K)
healthy_pred89(41-50,asympt,109-138,normal,no,0-2.1,up,color1,normal)

A.9 The Breast cancer problem domain

- Target predicate: benign(clump-thick, cell-size, cell-shape, margin, epith-size, nuclei, chrom, nucleoli, mitoses)

  - Rules extracted using PBT-RuleVI

benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred0(D,E,F,H,I)
benign_pred0(5,7,10,2,1)

benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred1(C,E,I)
benign_pred1(1,2,1)

benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred2(A,C,I)
benign_pred2(6,8,1)

$\neg$benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred3(C,F)
benign_pred3(8,9)

benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred4(A,E,F,I)
benign_pred4(1,2,1,1)

$\neg$benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred5(A,E,F)
benign_pred5(10,2,8)

$\neg$benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred6(C,D,E)
benign_pred6(3,4,2)

benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred38(A,B,C,E,H,I)
benign_pred38(4,6,5,7,9,1)

$\neg$benign(A,B,C,D,E,F,G,H,I) $\Leftarrow$ benign_pred41(D,F)

benign_pred41(10,1)

A.10 The Queensland Rail problem domain

- Target predicate: risky-track(type, protect, traffic_light, track, surface, speed, road-visible, pedestr, approach-signs)

  - Rules extracted using PCC-RuleVI

risky-track(A,B,C,D,E,F,G,H,I) $\Leftarrow$ risky-track_pred0(G)
risky-track_pred0(poor)

risky-track(A,B,C,D,E,F,G,H,I) $\Leftarrow$ risky-track_pred1(F, H)
risky-track_pred1(fast,medium)

risky-track(A,B,C,D,E,F,G,H,I) $\Leftarrow$ risky-track_pred2(E)
risky-track_pred2(concrete),risky-track_pred2(dirt),risky-track_pred2(gravel)

risky-track(A,B,C,D,E,F,G,H,I) $\Leftarrow$ risky-track_pred3(D)
risky-track_pred3(3ormore)

$\neg$risky-track(A,B,C,D,E,F,G,H,I) $\Leftarrow$ risky-track_pred4(B, H)
risky-track_pred4(boomgate,nil), risky-track_pred4(close,nil),
risky-track_pred4(flash,nil), risky-track_pred4(fence,nil)

$\neg$risky-track(A,B,C,D,E,F,G,H,I) $\Leftarrow$ risky-track_pred5(A)
risky-track_pred5(occup), risky-track_pred5(pedest),
risky-track_pred5(qr), risky-track_pred5(remove)

Bibliography

[Alexander, 1994] J. A. Alexander. *Template-Based Procedures for Neural Network Interpretation.* Masters Thesis, 1994.

[Andrews and Geva, 1994] R. Andrews and S. Geva. Rule extraction from a constrained error back propagation mlp. In *Proc. of 5th Australian Conference on Neural Networks, Brisbane, Australia*, pages 9–12, 1994.

[Andrews and Geva, 1996] R. Andrews and S. Geva. Rules and local function networks. In R. Andrews and J. Diederich, editors, *Rule and Networks*, Society For the Study of Artificial Intelligence and Simulation of Behavior Workshop Series (AISB/32596), pages 1–15, 1996.

[Andrews *et al.*, 1995] R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8:373–389, 1995.

[Andrews *et al.*, 1996] R. Andrews, R. Cable, J. Diederich, S. Geva, M. Golea, R. Hayward, C Ho-Stuart, and A. B. Tickle. An evaluation and comparison of techniques for extracting and refining rules from artificial neural networks. Technical Report QUT-NRC-96-01, QUT Internal Report, 1996.

[Angluin, 1992] D. Angluin. Computational learning theory: survey and selected bibliography. In *Proc. 24th Annual ACM Symposium on Theory Computation*, pages 351–369, NewYork, NY, 1992. ACM Press.

[Anthony and Biggs, 1995] M. Anthony and N. Biggs, editors. *Computational Learning Theory.* Boston: Kluwer Academic, 1995.

[Anthony, 1997] M. Anthony. Probabilistic analysis of learning in artificial neural networks: The pac model and its variants. In *Neural Computing Surveys*, volume 1, pages 1–47, 1997.

[Atlas *et al.*, 1989] L. Atlas, R. Cole, J. Connor, M. El-Sharkawi, R. Marks, and Y. Muthusamy. Performance of comparison between backpropagation networks and classification trees on three real-world applications. In D. Touretzky, editor, *Advances in Neural Information Processing systems*, volume 2. Morgan-Kaufmann, 1989.

[Barnden and Srinivas, 1991] J. A. Barnden and K. Srinivas. Encoding techniques for complex information structures in connectionist system. *Connection Science*, 3(3):263–309, 1991.

[Bechtel and Abrahamsen, 1991] W. Bechtel and A. Abrahamsen. *Connectionism and the mind: An introduction to parallel processing in networks.* Blackwell: Oxford UK and Cambridge USA, 1991.

[Bergadano and Gunetti, 1995] F. Bergadano and D. Gunetti. *Inductive Logic Programming: From Machine Learning to Software Engineering.* The MIT Press, Cambridge, 1995.

[Blassig, 1994] R. Blassig. Gds: Gradient descent generation of symbolic rules. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in neural information processing systems*, volume 6, pages 1093–1100. Morgan Kaufmann, 1994.

[Blumer *et al.*, 1989] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[Boose, 1991] J. H. Boose. Knowledge acquisition tools, methods and mediating representations. In H. Motoda, P. Mizoguchi, J. Boose, and B. Gaines, editors, *Knowledge acquisition for Knowledge based systems.* Amsterdam: IOS press, 1991.

[Brachman *et al.*, 1989] R. J. Brachman, H. J. Levesque, and R. Reiter, editors. *Proceedings of first international conference on principles of knowledge representation and reasoning.* CA: M. Kaufmann, 1989.

[Breiman *et al.*, 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees.* Belmont, CA : Wadsworth, 1984.

[Breiman, 1996] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Browne, 1997] A. Browne, editor. *Neural network analysis, architectures, and applications.* Philadelphia : Institute of Physics, 1997.

[Buchanan and Shortliff, 1984] B. G. Buchanan and E. H. Shortliff, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley, 1984.

[Bundy, 1980] A. Bundy. *Artificial Intelligence: An Introductory Course.* Edinburgh University Press, Edinburgh, 1980.

[Buntine, 1988] W. Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36:149–176, 1988.

[Cabena *et al.*, 1997] P. Cabena, P. Hadjinian, R. Stadler, J. Verhees, and A. Zanasi. *Discovering data mining: From concept to implementation.* Prentice Hall PTR, N.J., 1997.

[Castellanos *et al.*, 1997] A L. Castellanos, J. Castellanos, J. Manrique, and A. Martinez. A new approach for extracting rules from a trained neural network. In *Progress in Artificial Intelligence*, pages 297–302. Springer, 1997.

[Catlett, 1991] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *Proc. of the EWSL-91*, pages 164–177, 1991.

[Cestnik *et al.*, 1987] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: A knowledge elicitation tool for sophisticated users. In I. Bratko and L. Lavrac, editors, *Progress in machine learning.* Wilmslow: Sigma Press, 1987.

[Cios *et al.*, 1998] K. J. Cios, W. Pedrycz, and R. W. Swiniarsk. *Data mining methods for knowledge discovery.* Boston : Kluwer Academic, c1998, 1998.

[Clark and Niblett, 1989] P. Clark and R. Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–284, 1989.

[Collier and Waugh, 1994] P. A. Collier and S. G. Waugh. Characteristics of data suitable for learning with connectionist and symbolic methods. In C. Zhang, J. Debenham, and D. Lukose, editors, *Proc. of 7th Australian Joint Conference on Artificial Intelligence, Singapore*, pages 116–123, 1994.

[Cottrell, 1990] W.G. Cottrell. Extracting features from faces using compression networks: Face, identity, emotion and gender recognition using holons. In *Connection Models: Proc. of the 1990 Summer School.* San Mateo, Morgan Kaufmann, 1990.

[Craven and Shavlik, 1992] M. Craven and J. W. Shavlik. Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools*, 1(2):399–425, 1992.

[Craven and Shavlik, 1993] M. Craven and J. W. Shavlik. Learning symbolic rules using artificial neural networks. In *Proc. 10th International conference on Machine Learning, Amherst, MA*, pages 73–80. Morgan Kaufmann, LosAltos, CA, 1993.

[Craven and Shavlik, 1994] M. Craven and J. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning: Proc. of the Eleventh International Conference, San Francisco, CA, USA*, 1994.

[Craven and Shavlik, 1997] M. Craven and J. W. Shavlik. Using neural networks for data mining. *Future Generation Computer Systems*, 13:211–229, 1997.

[Craven, 1996] M. W. Craven. *Extracting comprehensible models from trained neural networks.* PhD thesis, University of Wisconsin, 1996.

[Datta, 1993] S. Datta. *Knowledge processing and applied artificial intelligence.* Oxford: Butterworth-Heinemann, 1993.

[de Mantaras and Armengol, 1998] R. Lopez de Mantaras and E. Armengol. Machine learning from examples: Inductive and lazy methods. *Data and Knowledge Engineering*, 25:99–123, 1998.

[Dejong and Mooney, 1986] G. Dejong and R. Mooney. Explanation based learning: An alternative view. *Machine Learning*, 1:145–176, 1986.

[Diederich *et al.*, 1998] J. Diederich, R. Nayak, and R. Hayward. From inductive to analytical neural network learning. Technical report, MLRC, QUT, Brisbane, Australia, 1998.

[Diederich, 1991] J. Diederich. Re-learning in connectionist semantic networks. In K. Morik, F. Bergadano, and W. Buntine, editors, *IJCAI-91 Workshop: Evaluation and Changing Representation in Machine Learning*. Sydney, Australia, 1991.

[Diederich, 1992] J. Diederich. Explanation and artificial neural networks. *International journal of Man Machine Studies*, 37:335–355, 1992.

[Dietterich and Michalski, 1981] T. G. Dietterich and R. S. Michalski. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 16:257–294, 1981.

[Dillon *et al.*, 1997] T. Dillon, P. Arabshahi, and R. J. Marks, editors. *A special issue on everyday applications of NNs*, volume 8 of *IEEE Trans. on NNs*, July 1997.

[Dillon, 1997] L. Dillon. *An ANN Learning System for SHRUTI.* Honours Thesis, 1997.

[Dorffner, 1997] G. Dorffner, editor. *Neural networks and a New Artificial Intelligence.* International Thomson Computer Press, 1997.

[Dougherty *et al.*, 1995] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. *Proc. of 12 th international conference on machine learning*, pages 194–202, 1995.

[Dreyfus and Dreyfus, 1986] H. L. Dreyfus and S. E. Dreyfus. *Mind over Machine: The power of human intuition and expertise in the era of the computer.* New York: The Free Press, 1986.

[Edmond, 1992] D. Edmond. *Information Modeling: Specification and Implementation.* Prentice Hall, 1992.

[Fahlman and Lebiere, 1990] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, volume 2. Morgan Kaufmann, 1990.

[Fahlman, 1979] S. E. Fahlman. *NETL: A System for Representing Real-world Knowledge.* MIT Press, Cambridge MA, 1979.

[Fahlman, 1988] S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proc. of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, 1988.

[Fausett, 1994] L. Fausett. *Fundamentals of Neural networks.* Prentice Hall, 1994.

[Fayyad and Irani, 1993] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th IJCAI*, 1993.

[Fayyad, 1996] U. Fayyad. Data mining and knowledge discovery. *IEEE Expert*, 11(5):20–25, 1996.

[Feldman and Ballard, 1982] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6(3):205–254, 1982.

[Feldman *et al.*, 1988] J. A. Feldman, M. A. Fanty, and N. H. Goddard. Computing with structured neural networks. *IEEE Computers*, pages 91–103, 1988.

[Feldman, 1982] J. A. Feldman. Dynamic connections in neural networks. *Biological Cybernetics*, 46:27–39, 1982.

[Feldman, 1986] J. A. Feldman. Neural representation of conceptual knowledge. Technical Report 189, Computer Science Department, University of Rochester, 1986.

[Firebaugh, 1989] M. W. Firebaugh. *Artificial Intelligence: A Knowledge Based Approach.* PWS-Kent Publishing Company, Boston, 1989.

[Fletcher and Hinde, 1995] G. P. Fletcher and C. J. Hinde. Using neural networks as a tool for constructing rule based systems. *Knowledge-Based Systems*, 8(4):183–189, 1995.

[Fodor and Z. W, 1988] J. A. Fodor and Pylyshyn Z. W. Connectionism and cognitive architecture: A critical analysis. In S. Pinker and J. Mehler, editors, *Connectionism and Symbols*, pages 3–72. Elsevier science Publishers, 1988.

[Freund and Shapire, 1997] Y. Freund and R. E. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[Friedman, 1977] J. H. Friedman. A recursive partitioning decision rules for non-parametric classification. In *IEEE Transactions on Computers*, pages 404–408, 1977.

[Fu, 1989] L. M. Fu. Integration of neural heuristics into knowledge-based inference. *Connection Science*, 1(3):325–339, 1989.

[Fu, 1994] L. M. Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man and Cybernetics*, 24(8):1114–1124, 1994.

[Fu, 1995] L. M. Fu. Introduction to knowledge-based neural networks. *Knowledge-based Systems*, 8(6):279–300, 1995.

[Furnkranz and Pfahringer, 1998] J. Furnkranz and B. Pfahringer, editors. *A special issue on first order knowledge discovery in databases*, volume 12 of *Applied Artificial Intelligence*, July-August 1998.

[Gallant, 1990] S. I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

[Gallant, 1993] S. I. Gallant. *Neural Network Learning and Expert Systems.* MIT Press, Cambridge, MA, 1993.

[Garey and Johnson, 1979] M. Garey and D. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: S. H. Freeman, 1979.

[Geva and Orlowski, 1996] S. Geva and M. Orlowski. On identification of decision rules with functional dependency preprocessing. In *Proc. of 4th European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany*, pages 1600–1604, 1996.

[Geva and Sitte, 1992] S. Geva and J. Sitte. A constructive method of multivariate function approximation by multilayer perceptron. *IEEE Transaction on Neural Networks*, 1992.

[Gűsgen and Hőlldobler, 1992] H. W. Gűsgen and S. Hőlldobler. Connectionist inference systems. In B. Fronhofer and G. Wrightson, editors, *Parallelisation in Inference systems.* Lecture Notes in Artificial Intelligence, Springer, Berlin, 1992.

[Golea, 1996] M. Golea. On the complexity of rule extraction from neural networks and network querying. In R. Andrews and J. Diederich, editors, *Rule and Networks*, Society For the Study of Artificial Intelligence and Simulation of Behavior Workshop Series (AISB/32596), pages 51–59, 1996.

[Gorman and Sejnowski, 1988] R. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[Gottlob, 1987] G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24:109–111, 1987.

[Grassmann and Tremblay, 1998] W. K. Grassmann and J. P. Tremblay. *Logic and Discrete Mathematics: A computer science perspective.* Prentice Hall, 1998.

[Hagiwara, 1993] M. Hagiwara. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6:207–218, 1993.

[Hammadi and Korczak, 1995] M.F. Hammadi and J.J. Korczak. An unsupervised neural network classifier and its application in remote sensing. Technical report, University Louis Pasteur, Stransbourg, France, 1995.

[Hanson and Burr, 1990] S. Hanson and D. Burr. What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Science*, 13:471–518, 1990.

[Hartigan, 1975] J. A. Hartigan. *Clustering algorithms.* New York: Wiley, 1975.

[Hassibi and Stork, 1994] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brian surgeon,. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in neural information processing systems*, volume 6, pages 164–171. San Mateo, CA: Morgan Kaufmann, 1994.

[Haussler, 1988] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4:7–40, 1988.

[Hayward *et al.*, 1996] R. Hayward, A. Tickle, and J. Diederich. Extracting rules for grammar recognition from cascade-2 networks. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Proc.*, pages 48–60. Springer-Verlag, Berlin, 1996.

[Hayward *et al.*, 1997] R. Hayward, C. Ho-Stuart, and J. Diederich. Neural networks as oracles for rule extraction. In *Connectionist System for Knowledge Representation and Deduction*, pages 105–116. Queensland University of Technology, Australia, 1997.

[Hayward, 1999] R. Hayward. Analytical learning in efficient connectionist rule-based reasoning system. Ph.D. Thesis in progress, 1999.

[Hőlldobler, 1990] S. Hőlldobler. A connectionist inference system for horn-logic based on the connection method and using limited resources. Technical Report TR-90-042, ICSI, Berkeley, 1990.

[Hilario, 1997] M. Hilario. Bias and knowledge in symbolic and connectionist induction. Technical Report UNIGE-97-03, CUI, University of Geneva, 1997.

[Hinton, 1986] G. E. Hinton. Learning distributed representations of concepts. In *Proc. of the 8th Annual conference of the Cognitive Science Society, Amherst, MA*, pages 1–12, 1986.

[Hornik *et al.*, 1989] K. Hornik, M. Stinchcombe, and H. White. Multi-layer feedforward neural networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[Imam and Michalski, 1993] I. F. Imam and R. S. Michalski. Should decision trees be learned from examples or from decision rules? In J. Komorowski and Z. W. Ras, editors, *Methodologies for Intelligent Systems*, pages 395–405. Springer-Verlag, 1993.

[Ishikawa, 1996] M. Ishikawa. Structural learning with forgetting. *Neural Networks*, 9(3):509–521, 1996.

[Kalinke, 1997] Y. Kalinke. Using connectionist term representations for first-order deduction- a critical view. In *Connectionist System for Knowledge Representation and Deduction*, pages 36–42. Queensland University of Technology, Australia, 1997.

[Kearns and Vazirani, 1994] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, 1994.

[Kerber, 1992] R. Kerber. Chimerge: Discretisation of numeric attributes. In *Proc. of the 10th National conference on Artificial Intelligence*, pages 123–128. AAAI/MIT Press, 1992.

[Lallement and Alexandre, 1997] Y. Lallement and F. Alexandre. Cognitive aspects of neurosymbolic integration. In R. Sun and F. Alexandre, editors, *Connectionist Symbolic Integration*, pages 57–68. Lawrence Erlbaum Associates, 1997.

[Lang *et al.*, 1990] J. Lang, H. Waibel, and E. G. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:33–43, 1990.

[Lange and Dyer, 1989] T. E. Lange and M. G. Dyer. High-level inferencing in a connectionist network. *Connection Science*, 1(2):181–217, 1989.

[Lapedea and Faber, 1987] A. Lapedea and R. Faber. How neural nets work. In *Neural Information Processing systems*, pages 442–456, Denver, 1987. American Institute of Physics, New York.

[Lavrac and Dzeroski, 1994] N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications.* Horwood Series in Artificial Intelligence. Ellis Horwood, London, 1994.

[Lavrac *et al.*, 1991] N. Lavrac, S. Dzeroski, and M. Grobelnick. Learning non recursive definitions of relations with linus. In Y. Kodratoff, editor, *Machine Learning-EWSL'91*, pages 265–281, 1991.

[Lecun *et al.*, 1989] Y. Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(14), 1989.

[Lin and Hendler, 1995] C. Lin and J. Hendler. Examining a hybrid connectionts/symbolic system for the analysis of ballistic signals. In R. Sun and L. A. Bookman, editors, *Computational Architectures Integrating neural and Symbolic Processes*, pages 319–348. Kluwer Academic Publishers, 1995.

[Linsay *et al.*, 1980] R. K. Linsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg. *Application of artificial intelligence for organic chemistry: the DENDRAL project.* NewYork:McGraw-Hill, 1980.

[Liu and Tan, 1995] H. Liu and S.T. Tan. X2r: A fast rule generator. In *IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada*, pages 1631–1635, 1995.

[Lloyd, 1987] J. W. Lloyd. *Foundation of Logic Programming.* Berlin, New York: Springer Verlag, 1987.

[Maire, 1999] F. Maire. Rule-extraction by backpropagation of polyhedra. publication in process, 1999.

[Mani, 1995] D. R. Mani. *The Design and Implementation of Massively Parallel Knowledge Representation and Reasoning Systems: A Connectionist Approach.* PhD thesis, University of Pennsylvania, 1995.

[Marcinkowski and Pacholski, 1992] J. Marcinkowski and L. P. Pacholski. Undecidability of the horn clause implication problem. In *Proceedings of the 33rd IEEE .. on Foundations of Computer Science*, pages 354–362. IEEE Comp Soc Press, Pittsburgh, PA, 1992.

[McMillan *et al.*, 1991] C McMillan, C Mozer, and P Smolensky. The connectionist scientist game: rule extraction and refinement in a neural network. In *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ*, 1991.

[Melnik and Pollack, 1999] O. Melnik and J. B. Pollack. Exact representations from feed-forward networks. Technical Report CS-99-205, Computer Science Department, Brandeis University, 1999.

[Michalski and Chilausky, 1980] R. S. Michalski and R. L. Chilausky. Knowledge acquisition by encoding expert rules versus computer induction from examples-a case study involving soya-bean pathology. *International Journal of Man-Machine Studies*, 12:63–87, 1980.

[Michalski, 1980] R. S. Michalski. Knowledge acquisition through conceptual clustering: a theoretical framework and an algorithm for partitioning data into conjunctive concepts. *International J. Policy, Anal. Inform. Systems*, 4(3):219–244, 1980.

[Michalski, 1983] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[Michie *et al.*, 1994] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Hertfordshire : Ellis Horwood, 1994.

[Miller *et al.*, 1990] W. T. Miller, R. S. Sutton, and P. J. Werbos, editors. *Neural networks for control.* Cambridge: MIT Press, 1990.

[Minsky, 1990] M. Minsky. Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. In P. H. Winston, editor, *Artificial Intelligence at MIT., Expanding Frontiers.* MIT Press, 1990.

[Mitchell, 1980] T. M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of computer science, Rutgers University, New Brunswick, NJ., 1980.

[Mitchell, 1982] T. M. Mitchell. Generalisation as search. *Machine Learning*, 18(2):203–225, 1982.

[Mitchell, 1997] T. M. Mitchell. *Machine Learning.* The McGraw-Hill Companies, Inc, 1997.

[Moldovan *et al.*, 1992] D. I. Moldovan, W. Lee, C. Lee, and M. Chung. Snap: Parallel processing applied to ai. *Computer*, 25(5):39–50, 1992.

[Mozetic, 1985] I. Mozetic. Program for learning from examples - technical documentation and user's guide. Technical Report IJS-DP-4390, Jozef Stefan Institute, Ljubljana, 1985.

[Muggleton and Buntine, 1988] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In A. Arbor, editor, *Proc. of the 5th international conference on Machine Learning*, pages 339–352. Morgan Kaufmann, 1988.

[Muggleton and DeRaedt, 1994] S. Muggleton and L. DeRaedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[Muggleton and Feng, 1990] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. of the 5th conference on Algorithmic Learning Theory.* OHMSHA, Tokyo, 1990.

[Muggleton, 1990] S. Muggleton. *Inductive Acquisition of Expert Knowledge.* Turing Institute Press and Addison-Wesley, 1990.

[Muggleton, 1991] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[Muggleton, 1995] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.

[Murphy, 1995] P. Murphy. The uc irvine repository of machine learning databases. Available from: http://www.ics.uci.edu/ mlearn/MLSummary.html, 1995.

[Murray, 1992] A. F. Murray, editor. *Applications of neural networks.* Cambridge Tracts in Theoretical Computer Science (30). Cambridge University Press, 1992.

[Nayak and Diederich, 1998] R. Nayak and J. Diederich. A knowledge representation formalism generation from the feedforward neural networks. In *Proceedings of the International conference on advance computing, IEEE computer society*, pages 255–264, 1998.

[Nayak and Sharma, 1999] R. Nayak and J. D. Sharma. A hybrid intelligent approach to solve unit-commitment problem. forthcoming in International Journal of Computers and Electrical Engineering, 1999.

[Nayak *et al.*, 1997] R. Nayak, R. Hayward, and J. Diederich. Connectionist knowledge representation by extracted generic rules from trained feedforward neural networks. In *Connectionist System for Knowledge Representation and Deduction*, pages 87–97. Queensland University of Technology, Australia, 1997.

[Nayak *et al.*, 1999] R. Nayak, J. Diederich, and F. Maire. Inductive knowledge acquisition using feedforward neural network. In *Proceedings of Pacific Rim knowledge acquisition workshop*, pages 74–86, 1999.

[Nayak, 1996] R. Nayak. Rule-insertion in a connectionist semantic network generated by a artificial neural network. Technical Report QUT-ITN-447 (Semester Report), NRC, QUT, Brisbane, Australia, 1996.

[Orlowski, 1992] M. Orlowski. An algorithm for the maintenance of functional relationships. In W. Koczkodaj, editor, *Computing and Information*, pages 377–380. IEEE Computer Society Press, 1992.

[Orlowski, 1995] M. Orlowski. On the integration of knowledge bases. *Proceedings of the VI International Conference on Systems Research*, pages 483–487, 1995.

[Park *et al.*, 1995] N. S. Park, D. Robertson, and K. Stenning. An extension of the temporal synchrony approach to dynamic variable binding in a connectionist inference system. *Knowledge Based Systems*, 8(6):345–358, 1995.

[Pazzani and Kibler, 1992] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94, 1992.

[Perthold and Hand, 1999] M. Perthold and D. J. Hand, editors. *Intelligent data analysis: An introduction.* Berlin: Springer Verlag, 1999.

[Pinkas, 1991] G. Pinkas. Symmetric neural networks and propositional logic. *Neural computation*, 3:282–291, 1991.

[Pinker and Prince, 1988] S. Pinker and A. Prince. On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. In S. Pinker and J. Mehler, editors, *Connectionism and Symbols*, pages 73–194. Elsevier science Publishers, 1988.

[Plotkin, 1970] D. G. Plotkin. A note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[Plotkin, 1971] D. G. Plotkin. A further note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, volume 6, pages 101–124. Edinburgh University Press, 1971.

[Pop *et al.*, 1995] E. Pop, R. Hayward, and J. Diederich. Ruleneg: Extracting rules from a trained neural network by step-wise negation. In *Proc. of the Third Conference of the Australian Cognitive Science Society*, pages 62–69, April 1995.

[Popplestone, 1970] R. J. Popplestone. An experiment in automatic induction. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 203–215. Edinburgh University Press, 1970.

[Pratt *et al.*, 1991] L. Pratt, J. Mostow, and C. Kamm. Direct transfer of learned information among neural networks. In *Proc. of the 9th National conference on Artificial Intelligence, Anaheim, CA*, pages 584–589. AAAI/MIT Press, 1991.

[Prechelt, 1995] L. Prechelt. Adaptive parameter pruning in neural networks. Technical Report TR-95-009, International Computer Science Institute, Berkeley, California, 1995.

[Quinlan and Cameron-Jones, 1995] J. R. Quinlan and M. Cameron-Jones. Induction of logic programs: Foil and related systems. *New Generation Computing*, 13:287–312, 1995.

[Quinlan, 1979] J. R. Quinlan. Induction over large databases. Technical Report HPP-79-14, Heuristic Programming Project, Stanford University, 1979.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1990] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[Quinlan, 1993a] J. R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann Publishers, 1993.

[Quinlan, 1993b] J. R. Quinlan. Comparing connectionist and symbolic learning methods. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*, pages 445–456. Cambridge, MA: MIT Press, 1993.

[Quinlan, 1994] J. R. Quinlan. The minimum description length principle and categorical theories. In *Proc. of the 11th international conference on Machine Learning*, pages 233–241. San Fansico:Morgan Kaufmann, 1994.

[Quinlan, 1996] J. R. Quinlan. Bagging, boosting and c4.5. In *Proc. AAAI-96 14th National Conference on Artificial Intelligence.* AAAI Press, Menlo Park, CA, 1996.

[Quinlan, 1998] J. R. Quinlan. C5, 1998. http://www.rulequest.com.

[Ramsay, 1988] A. Ramsay. *Formal Methods in Artificial Intelligence.* Cambridge University Press, 1988.

[Rissanen, 1983] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11:416–431, 1983.

[Rojas, 1996] R. Rojas. *Neural networks: A Systematic Introduction.* Springer, 1996.

[Rouveirol and Puget, 1989] C. Rouveirol and J. F. Puget. Beyond inversion of resolution. In *Proc. of Sixth international workshop on Machine Learning*, pages 122–130, San Mateo, CA, USA, 1989. Morgan Kaufmann.

[Rumelhart *et al.*, 1986] D.E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel distributed processing (Vol. 1. Foundations)*. Cambridge, MA:MIT Press, 1986.

[Rumelhart, 1990] D.E. Rumelhart. Brain style computation: Learning and generalisation. In Zornetzer, editor, *An Introduction to Neural and Electronic Networks*, pages 405–420. Academic Press, New York, 1990.

[Russell and Norvig, 1995] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach.* Prentice Hall, NJ, 1995.

[Russell, 1996] S. Russell. Machine learning. In M. A. Boden, editor, *Artificial Intelligence*, pages 90–113. Academic Press, NY, USA, 1996.

[Saito and Nakano, 1988] K. Saito and R. Nakano. Medical diagnostic expert system based on pdp model. In *Proc. of the IEEE International conference on Neural Networks, San Diego, CA*, pages 255–262, 1988.

[Sanger, 1989] D. Sanger. Contribution analysis: A technique for assigning responsibilities to hidden units in connectionist networks. *Connection Science*, 1(2):115–138, 1989.

[Sestito and Dillon, 1994] S. Sestito and T. Dillon. *Automated Knowledge Acquisition.* Englewood Cliffs, NJ: Prentice Hall, 1994.

[Sethi and Yoo, 1994] I. Sethi and J Yoo. Symbolic approximation of feedforward neural networks. In *Patterns Recognition in Practice IV*, pages 313–324. North-Holland, 1994.

[Setiono and Leow, 1999] R. Setiono and W.K. Leow. On mapping decision trees and neural networks. forthcoming in Knowledge Based Systems, 1999.

[Setiono and Liu, 1996] R. Setiono and H. Liu. Symbolic representation of neural networks. *IEEE Computer*, pages 71–77, March 1996.

[Setiono and Liu, 1997] R. Setiono and H. Liu. Neurolinear: from neural networks to oblique decision rules. *Neurocomputing*, 17(1):1–25, 1997.

[Setiono, 1997a] R. Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9(1):205–225, 1997.

[Setiono, 1997b] R. Setiono. A penalty function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–205, 1997.

[Shapiro, 1983] E. Y. Shapiro. *Algorithmic Program Debugging.* The MIT Press, 1983.

[Shastri and Ajjanagadde, 1993] L. Shastri and V. Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings. *Behavioral and Brain Science*, 16:417–494, 1993.

[Shastri, 1988] L. Shastri. *Semantic Networks: An evidential formulation and its connectionist realization.* Pitman/Morgan Kaufman, 1988.

[Shastri, 1990] L. Shastri. Connectionism and the computational effectiveness of reasoning. *Theoretical Linguistics*, 16(1):65–87, 1990.

[Shastri, 1991] L. Shastri. Relevance of connectionism to ai: A representation and reasoning perspective. In J. Barnden and J. B. Pollack, editors, *Advances in connectionist and neural computational theory, vol 1*, pages 259–283. Ablex, Norwood, NJ, 1991.

[Shastri, 1992] L. Shastri. Structured connectionist models of semantic networks. *Computers Math. Application*, 23(2-5):293–328, 1992.

[Shastri, 1999] L. Shastri. Advances in shruti - a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. forthcoming in Applied Intelligence, 1999.

[Shavlik and Dietterich, 1990] J. W. Shavlik and T. G. Dietterich, editors. *Readings in machine learning.* Morgan Kaufmann Publishers, 1990.

[Shavlik *et al.*, 1991] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: an experimental comparison. *Machine Learning*, 6:111–143, 1991.

[Simpson, 1996] P. K. Simpson, editor. *Neural networks applications.* New York: Institute of Electrical and Electronics Engineers, 1996.

[Sporring, 1995] J. Sporring. Pruning with minimum description length. In *Proc. of 5th Scandinavian Symposium on Artificial Intelligence*, pages 157–168. IOS Press, Amsterdam, Netherlands, 1995.

[Srinivasan and King, 1996] A. Srinivasan and R. D. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity by structural attributes. In S. Muggleton, editor, *Inductive Logic Programming*, pages 89–104. Springer-Verlag, 1996.

[Sun and Bookman, 1995] R. Sun and L. A. Bookman, editors. *Computational Architectures Integrating Neural and Symbolic Processes.* Kluwer Academic Publishers, 1995.

[Sun and Peterson, 1997] R. Sun and T. Peterson. A hybrid agent architecture for the reactive sequential decision making. In R. Sun and F. Alexandre, editors, *Connectionist Symbolic Integration*, pages 113–138. Lawrence Erlbaum Associates, 1997.

[Sun, 1994] R. Sun, editor. *Integrating rules and connectionism for robust commonsense reasoning.* John Wiley and Sons, 1994.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning.* The MIT Press, 1998.

[Taha and Ghosh, 1996] I. Taha and J. Ghosh. Three techniques for extracting rules from feedforward networks. In A. Dagli, F. Chen, and J. Ghosh, editors, *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 6, pages 23–28. ASME Press, St. Louis, 1996.

[Taha and Ghosh, 1997] I. Taha and J. Ghosh. Evaluation and ordering of rules extracted from feedforward networks. In *Proc. of the IEEE international conference on NNs, Houston, Texas*, pages 408–413, 1997.

[Thomas, 1989] R. Thomas. *Clausal from logic: an introduction to the logic of computer reasoning.* Addison-Wesley, 1989.

[Thornton, 1992] C. J. Thornton. *Techniques in computational learning: An introduction.* CHAPMAN and Hall, 1992.

[Thrun *et al.*, 1991] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. CEstnik, J. Cheng, K. Dejong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalaski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, K. VandeWelde, W Wenzel, J. Wnek, and J. Zhang. The monk's problems: a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[Thrun, 1995] S. Thrun. Extracting rules from artificial neural networks with distributed representations. In *Advances in Neural Information Processing systems*, volume 7. The MIT Press, 1995.

[Tickle *et al.*, To appear] A. B. Tickle, F. Maire, G. Bologna, and J. Diederich. Lessons from the past, current issues, and future research directions in extracting the knowledge embedded with trained artificial neural network. In S. Wermter and R. Sun, editors, *Neural Hybrid Systems.* Springer-Verlag, To appear.

[Tickle, 1998] A. B. Tickle. *Machine Learning, neural networks and information security: techniques for extracting rules from trained feedforward neural networks.* PhD thesis, Queensland University of Technology, 1998.

[Touretzky and Hinton, 1988] D. S. Touretzky and G. E. Hinton. A distributed connectionist production system. *Cognitive Science*, 12(3):423–466, 1988.

[Towell and Shavlik, 1993] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[Tresp *et al.*, 1993] V Tresp, J. Hollatz, and S. Ahmad. Network structuring and training using rule-based knowledge. In S. Joshanson, J. D. Gowan, and C. L. Giles, editors, *Advances in Neural Information Processing systems*, volume 5, pages 71–101. Morgan-Kaufmann, 1993.

[Ulug, 1989] M. E. Ulug. A hybrid expert system combining ai techniques with a neural net. In *Proceedings of the second international conference on industrial and engineering applications of AI and expert systems*, pages 305–309, 1989.

[Valiant, 1984] L Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[Vere, 1975] S. Vere. Induction of concepts in the predicate calculus. In *Proc. of the fourth International joint conference on Artificial Intelligence*, pages 351–356, 1975.

[Viktor *et al.*, 1995] H.L. Viktor, A.P. Engelbrecht, and I Cloete. Reduction of symbolic rules from artificial neural networks using sensitivity analysis. In *IEEE conference on Neural Networks*, pages 1788–1793, November 1995.

[Visser *et al.*, 1998] U. Visser, R. Nayak, and M. T. Wong. Rule extraction from trained neural networks and connectionist knowledge representation for the determination of pesticide mixtures. In *Proceedings of the Ninth Australian conference on neural networks*, pages 138–142, 1998.

[Waltz and Pollack, 1985] D. L. Waltz and J. B. Pollack. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9:51–74, 1985.

[Waugh and Adams, 1993] S. G. Waugh and A. Adams. Comparison of inductive learning of classification tasks by neural networks. Technical Report TR-93-5, Department of Computer Science, University of Tasmania, 1993. An abstract appeared in the proceedings of AI'93.

[Weigend *et al.*, 1990] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. Technical Report P-90-00022, System Science Laboratory, Palo Alto Research Center, California, 1990.

[Weiss and Kapouleas, 1989] S. M. Weiss and I Kapouleas. An empirical comparison of patterns recognition, neural nets and machine learning methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 2, pages 688–693. Morgan Kaufmann, 1989.

[Weiss and Kulikowski, 1991] S. M. Weiss and C. A. Kulikowski. *Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning and expert systems.* Morgan Kaufmann Publishers, 1991.

[Widrow *et al.*, 1994] B. Widrow, D. E. Rumelhart, and M. A. Lehr. Neural networks: Applications in industry, business, and science. *Communications of the ACM*, 37(3):93–105, 1994.

[Winston, 1970] P. Winston. *Learning structural descriptions from examples.* PhD thesis, MIT, 1970. Also available as Technical Report, AI-TR-231.

[Wong, 1997] M. T. Wong. Experimental comparison of rule-extraction techniques from trained artificial neural networks and symbolic inductive inference. Technical report, QUT Internal Report, 1997.

[Wrobel, 1996] S. Wrobel. Inductive logic programming. In G. Brewka, editor, *Principles of Knowledge Representation.* CSLI Publications and FoLLI, 1996.

[Wu *et al.*, 1997] X. Wu, M. McTear, P. Ojha, and H. Dai. A hybrid system framework for disambiguating word senses. In R. Sun and F. Alexandre, editors, *Connectionist Symbolic Integration*, pages 227–243. Lawrence Erlbaum Associates, 1997.

[Yuanhui *et al.*, 1997] Z. Yuanhui, L. Yuchang, and S. Chunyi. Using learning and searching approach to explain neural network with distributed representations. In *Proc. of the IEEE international conference on Systems, Man and Cybernetics*, pages 1424–1429, 1997.